April 2013

Modular Environmental Controller for Sustainable Agriculture

Erik Richard Dahlinghaus
Worcester Polytechnic Institute

William Bryson Caproni
Worcester Polytechnic Institute

Modular Environmental Controller for Sustainable Agriculture

April 25, 2013

A Major Qualifying Project Report

Submitted to the Faculty of the

Worcester Polytechnic Institute

in partial fulfillment of the requirements for the

Degree of Bachelor of Science in

Electrical and Computer Engineering and Robotics Engineering

by:

William Bryson Caproni

Erik Richard Dahlinghaus

Project Number: MQP-SJB-4A12

Advisor: Stephen J. Bitar

Co-Advisor: Dr. Taskin Padir

We would like to acknowledge the help of some key people whose support was invaluable in the success of this project:

Our parents for supporting us throughout our education here at WPI and throughout our lives

Ruth and Al Caproni for sponsoring the development of our prototype

Robinson Levin who infrequently answered our questions, but always provided helpful guidance; or at least frustration

The people from the BeagleBone mailing list for providing all of the most valuable documentation

Marie Aoude for feeding us falafel to fuel our all night project frenzies

And of course, Professor Bitar for allowing us to choose our own project and take it in the direction we wanted; letting us push our limits, but keeping us grounded; and for dinner at the Sole

Last but not least, we would like to thank Fred Seymour for his continuted support throughout the project

Abstract

The objective of this project was to design and implement a modular control system for use in controlled environment agriculture. The system is comprised of a TI Cortex A-8 based master node which communicates with PIC18 based sensor and control nodes using the CAN bus standard over standard Ethernet cable. The network cable carries both serial communications and 24V power for nodes on the network for ease of installation and expandability. The system can be monitored and controlled via a dynamic web page. A proof of concept was built which accurately monitors CO_2 concentration, relative humidity, and temperature and controls standard outlets.

Contents

List of Figures

List of Tables

List of Equations

List of Abbreviations

BB	Beaglebone
BSP	Board Support Package
CAN	Controller Area Network
CEA	Controlled Environment Agriculture
EC	Electrochemical
ECAN	Enhanced Controller Area Network
GPIO	General purpose input-output
ICSP	In-circuit Serial Programmer
LCDC	LCD Controller
MSSP	Master Synchronous Serial Port
NBR	Nominal Bitrate
NDIR	Nondispersive infared detectors
NRZ	Non-Return to Zero
NTC	Negative Temperature Coefficient
OE	Open Embedded
ORM	Object Relational Mapper
OSI	Open Systems Interconnection
PIC	PIC18F25K80
RDBMS	Relational Database Management System
RH	Relative Humidity
RTD	Resistance Temperature Detector
SQL	Structured Query Language
TDS	Total Disolved Solids

Executive Summary

During World War I and World War II US citizens were encouraged to plant victory gardens at home to supplement the nation's food supply. Today the US is engaged in another war on the home front; the war for fresh nutritious fruits and vegetables. Chances are, most of the fruits and vegetables in your local supermarket are anything but local. Oftentimes produce is shipped thousands of miles, sometimes even from another continent, before it reaches the shelves in your market [25]. This long distance food system has led to produce with transportation survivability as a priority rather than quality.

The availability of high quality produce is an even greater issue in densely populated cities. Some urban areas have little access to affordable fresh food resources. These areas are known as food deserts and are categorically populated by lower income families [26]. It is clear that a different approach to food production and supply must be taken in order to provide sustainable, high quality produce for all [27].

The alternative to low quality shipped produce is to purchase locally sourced produce; however, locally sourced produce can be difficult if not impossible to obtain in urban environments which can span hundreds of miles. In some climates not all types of produce can be grown locally by traditional means. The advent of Controlled Environment Agriculture (CEA) offers a new horizon for local food production. The use of CEA can increase the amount and variety of locally grown produce. CEA is the use of greenhouses with tightly controlled environmental parameters, and is often times combined with some form of hydroponics. These greenhouses can be placed anywhere regardless of climate; with the use of artificial lighting they can even be indoors, opening new possibilities for farm locations. Large rooftops and unused parking lots are emerging as great locations for urban farms. Computer systems are almost always employed to control commercial CEA greenhouses. These systems are capable of controlling air temperature, relative humidity, CO_2 concentrations, nutrient dispersal, light timing, and ventilation without the need for human intervention. Additionally most systems provide logging capabilities which allow the farmer to gain a comprehensive understanding of all factors affecting plant growth [28]. However these commercial solutions for controlling greenhouses are prohibitively expensive for non-commercial operations. Instead smaller farmers often resort to using an array of different solutions such as individual timers and thermostats.

With public interest in local food gaining momentum there has been an explosion in the number of small commercial and hobbyist farms; many of which utilize some form of CEA. These new farms will demand affordable, capable computer control systems. This project aims to fill that void by providing an affordable, integrated solution for greenhouse monitoring and control which is modular and expandable. The proposed design allows the farmer to add new sensor and control nodes, forming multiple zones with different environmental conditions, using a single controller. This reduces the cost of expanding existing farms making it an attractive option for new commercial ventures.

Project Goals

The goal of this project is to design an expandable, modular greenhouse control system for use in controlled environment agriculture. The system will consist of a main controller and a robust network of powered sensors and controls. The system should be capable of monitoring and controlling environmental parameters including atmospheric CO_2 concentration, relative humidity, and temperature. Additionally the system should be capable of controlling the timing of pumps, lights, and fans, as well as monitoring water quality of a hydroponic system through the use of pH and electroconductivity probes. Interface with the system is provided through a touchscreen and a dynamic web application. Below is a list of fundamental design requirements:

1. 100% Stability
2. State resume in case of power failure
3. Up to 127 nodes
4. Touchscreen GUI
5. Web application for remote monitoring and configuration
6. Sensors connected and powered by one cable
7. Accurate sensors for measuring temperature, relative humidity and CO_2 concentration
8. Accurate sensors for monitoring pH and electroconductivity of hydroponic nutrient solutions
9. An interface for float switches for monitoring fluid levels
10. Output via switched outlets for ease of setup with appliances

Design Overview

A system level block diagram is shown in a Figure 1. Individual components, the design decisions leading to their selection, and how they interface with one another are covered in subsequent chapters. Additionally, a complete list of parts is available in Table 13 in Appendix A. A software block diagram is given in Figure 36 on page 58.

The master node houses the BeagleBone (BB). The BeagleBone communicates with sensor and control nodes using the CAN protocol over standard Ethernet cable. This cable carries serial data on one twisted pair, 24V power on two pairs; the final pair carries an additional ground and an active low reset signal which can be used to power-cycle all nodes on the network. Interface is offered by means of a dynamic web application hosted by the BeagleBone allowing remote administration and monitoring. Additionally a touchscreen screen offers direct interface with the system.

All nodes consist of a PIC18F25K80 microcontroller, a 24V to 5V power regulator, and a CAN transceiver. Sensor nodes additionally contain sensors for temperature, humidity and CO_2, or pH and Electroconductivity.

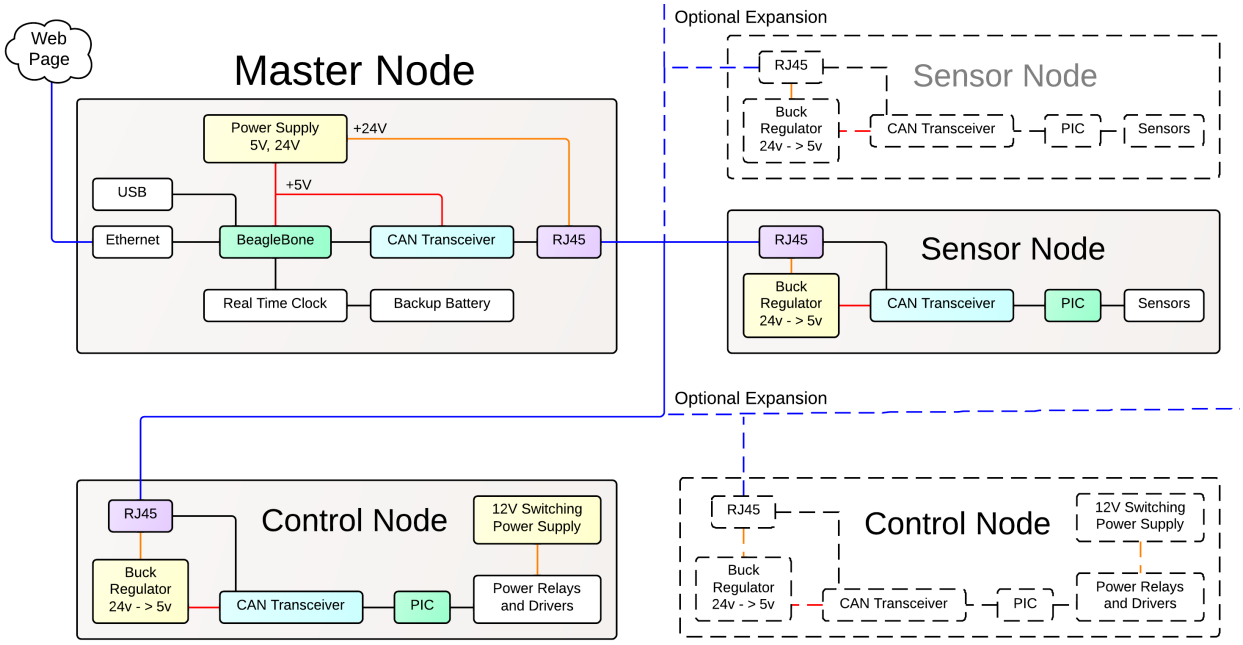

Figure 1: System Level Block Diagram

Control nodes contain a set of four power relays which control outlets, known as outputs, allowing interface with CO_2 generators, fans, lights, and pumps. The network is daisy-chained through nodes. Nodes can be assigned to zones allowing one master to control multiple independent environments.

Conclusions

It is clear that a drastic change to our food supply system must be made in order to make it more sustainable. Controlled environment agriculture may be the solution, but it will depend on the successful development of robust, effective control systems. Companies like PodPonics and The Farmery give us a glimpse into the potential future of agriculture, offering innovative solutions to the problems which face our food system today; however, low cost solutions are needed to reduce the barrier to entry into controlled environment agriculture. Inexpensive control systems like ours will pave the way, allowing companies and ideas like these to flourish.

We have demonstrated that a full featured greenhouse controller for controlled environment agriculture can be produced at a low enough price-point (around $400) to make it an attractive option for small startups and hobbyists. Our system is a practical alternative to existing solutions; the computer controlled system offers logging capabilites, remote access, and modularity allowing for future growth. While there are similarly featured products costing upwards of $3 000, most products at our price-point are severely lacking in functionality and often rely on mechanical timers and controls for operation.

We successfully completed six out of our original ten design goals; however, the unmet goals could

be easily implemented. Both the float switch interface and pH and electroconductivity sensors were left unimplemented due to time constraints, but could be easily integrated with minor additions to the existing node hardware and software. The touchscreen was successfully interfaced with the BeagleBone, however we were unable to successfully display a desired image. In our prototype phase we were able to achieve over 70% stability, an estimate taken from a five week period of testing. A summary of our design goal achievements can be found in Table 12. A photo our of final prototype can be found in Figure 2; the lightbulbs representing the controlled outputs.

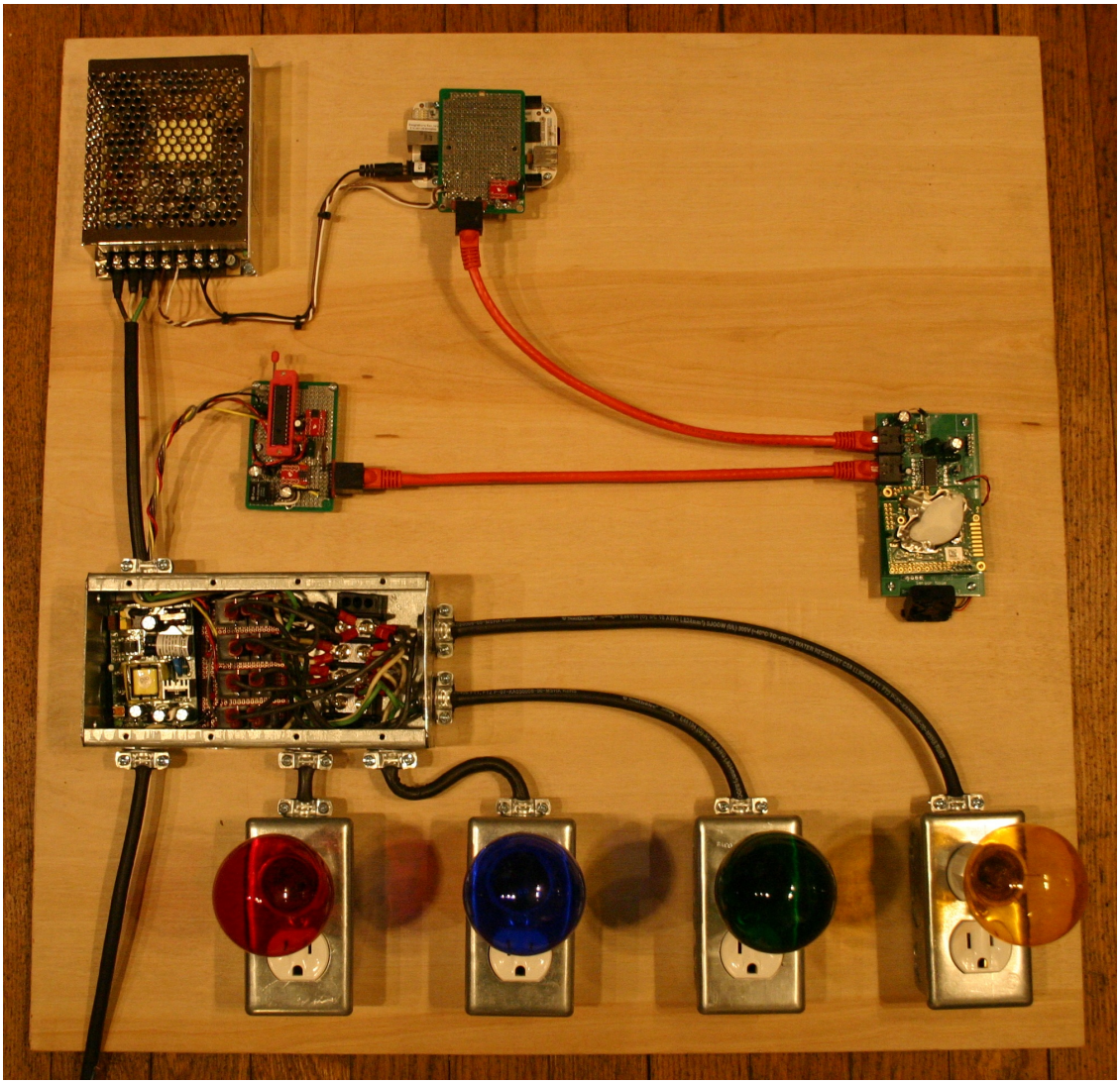

Figure 2: Final Prototype

1 Introduction

During World War I and World War II US citizens were encouraged to plant victory gardens at home to supplement the nation's food supply. Today the US is engaged in another war on the home front; the war for fresh nutritious fruits and vegetables. Chances are, most of the fruits and vegetables in your local supermarket are anything but local. Oftentimes produce is shipped thousands of miles, sometimes even from another continent, before it reaches the shelves in your market [25]. This long distance food system has led to produce with transportation survivability as a priority rather than quality.

The availability of high quality produce is an even greater issue in densely populated cities. Some urban areas have little access to affordable fresh food resources. These areas are known as food deserts and are categorically populated by lower income families [26]. It is clear that a different approach to food production and supply must be taken in order to provide sustainable, high quality produce for all [27].

The alternative to low quality shipped produce is to purchase locally sourced produce; however, locally sourced produce can be difficult if not impossible to obtain in urban environments which can span hundreds of miles. In some climates not all types of produce can be grown locally by traditional means. The advent of Controlled Environment Agriculture (CEA) offers a new horizon for local food production. The use of CEA can increase the amount and variety of locally grown produce. CEA is the use of greenhouses with tightly controlled environmental parameters, and is often times combined with some form of hydroponics. These greenhouses can be placed anywhere regardless of climate; with the use of artificial lighting they can even be indoors, opening new possibilities for farm locations. Large rooftops and unused parking lots are emerging as great locations for urban farms. Computer systems are almost always employed to control commercial CEA greenhouses. These systems are capable of controlling air temperature, relative humidity, CO_2 concentrations, nutrient dispersal, light timing, and ventilation without the need for human intervention. Additionally most systems provide logging capabilities which allow the farmer to gain a comprehensive understanding of all factors affecting plant growth [28]. However these commercial solutions for controlling greenhouses are prohibitively expensive for non-commercial operations. Instead smaller farmers often resort to using an array of different solutions such as individual timers and thermostats.

With public interest in local food gaining momentum there has been an explosion in the number of small commercial and hobbyist farms; many of which utilize some form of CEA. These new farms will demand affordable, capable computer control systems. This project aims to fill that void by providing an affordable, integrated solution for greenhouse monitoring and control which is modular and expandable. The proposed design allows the farmer to add new sensor and control nodes, forming multiple zones with different environmental conditions, using a single controller. This reduces the cost of expanding existing farms making it an attractive option for new commercial ventures.

The next chapter provides the reader with background information regarding different types of soil-less growing techniques; the current scales on which they operate; and justifications for using these techniques over conventional soil agriculture. It then introduces the reader to current research, commercial ventures, and existing products surrounding automated soil-less horticulture. Crucial to the success of accurate automation are accurate sensors; Chapter 3 describes the current available sensing technologies and their pertinence to

this project. Chapter 4 gives the reader knowledge about the CAN standard, and how it was implemented for this project. Chapters 5 and 6 go into detail of the two main components of the project: the BeagleBone, and the PIC. The BeagleBone is the master node, responsible for mediating communication between all other nodes on the network, as well as storing data and allowing user interface. All other nodes are comprised of a PIC microcontroller which communicates with the BeagleBone over the CAN network. In sensor nodes the PIC communicates with environmental sensors over I^2C; in the output nodes the PIC sets digital outputs which control relays. Finally, Chapter 7 concludes with a discussion of the goals attained, results observed, and recommendations for future work on the project. The appendices contain a complete list of parts used, the full schematics of the electronic circuitry, and access to the software package that was developed.

2 Background

2.1 Types of Soil-less Agriculture

In the early 18^{th} century researchers discovered that plants absorb essential mineral nutrients as inorganic ions in water. It was determined that soil itself is not necessary for plant growth, rather it acts only as a mineral nutrient reservoir. There are multiple techniques for growing plants without soil; the main three are hydroponics, aeroponics, and aquaponics. Hydroponics is the method of using an aqueous nutrient solution to grow plants in an inert medium such as perlite, gravel or mineral wool. This inert medium does not supply any nutrients to the plants, although much like soil it supplies an area for the roots to anchor and functions as a temporary reservoir for water and nutrients. Aeroponics does not depend on any growth medium, instead roots are suspended in air saturated with nutrient mist. Aquaponics is a food production system which combines hydroponics and aquaculture, the raising aquatic animals, typically fish. The fish waste provides food for the plants, and the plants clean the water for the fish to reuse.

Hydroponic

Hydroponics offers great energy and water savings compared to traditional agriculture. Hydroponics uses less energy than traditional farming because it doesn't require soil manipulation. Additionally, because the nutrients are contained in a closed system, none is lost to runoff or weeds, significantly reducing the cost of fertilizer. Other advantages to hydroponics include: stable plants with high yields, diseased and pest infested plants are more easily removed minimizing crop loss, and the use of a sterile growth medium reduces the need for pesticides or herbicides. However hydroponics is not without its disadvantages, without a soil buffer to hold water any failure to the hydroponic system will lead to swift crop loss. The medium in which the plants are grown must also be cleaned or replaced between crops.

A specific type of hydroponics, raft hydroponics, is often employed to grow lettuce. Heads of lettuce float on rafts which travel down a large hydroponic river; when they reach the end they are harvested.

Figure 3: Hydroponically Grown Lettuce [1]

Aeroponic

Aeroponics is a more recent extension of the hydroponic system that minimizes the resources needed to produce healthy plants. In aeroponics growing plants are supported so their roots are suspended in a chamber filled with nutrient mist. Aeroponics was originally developed in 1983 by Dr. Richard Stoner, who has been the principle investigator for several NASA aeroponic research grants [29]. An example of the setup is shown in figure Figure 4.

Figure 4: Aeroponic Plant Support Structure [2]

In hydroponic and soil based systems the medium can act as a vector allowing the spread of disease from plant to plant. In aeroponics there is no medium, therefore diseased plants can be removed and replaced with little or no spread to other plants. This allows for a more dense plant arrangement, faster growing, and healthier plants than those that can be grown using other systems. With the plants suspended in mid air more oxygen can get to the roots which stimulates root development and helps prevent disease. Additionally aeroponic systems consume less than 10% of the water used in conventional agriculture [30]. Aeroponics can also be designed to reclaim unused water and nutrients increasing efficiency. Without any growth medium at all, aeroponics even more than hydroponics, requires an advanced, robust control system to prevent losses in the event of equipment failure.

Aquaponic

Aquaponics is another horticultural method that mixes aquaculture and hydroponics. Here there are two symbiotic systems: one raising fish and the other growing plants. The fish generate ammonia which would accumulate over time proving toxic for the fish. The water is pumped through a biofilter which contains bacteria. These bacteria perform nitrification converting ammonia released by the fish into nitrates usable by the plants. The nitrates are then used by the plants, cleaning the water for the fish to reuse. This effectively creates a small functioning ecosystem with the only input being fish food.

As with the other systems aquaponics has its advantages and disadvantages. The organic fertilization of plants through the natural fish excretion and the complete recycling of water makes aquaponics highly sustainable. Additionally more fish can be raised in the same volume tank when compared to traditional aquaculture. Aquaponics presents additional concerns; the water must be adequately oxygenated to keep the fish alive, and the temperature must be maintained more carefully to prevent shock to the fish. An overview of the aquaponics cycle can be seen in Figure 5 on page 30.

2.2 Scale

The scales on which soil-less agriculture is performed are diverse, ranging from large-scale commercial farms to hobbyist window boxes. Large scale commercial farms produce vegetable and flower crops for exportation and domestic use, while small hobbyist systems may provide fresh herbs or produce to the grower.

Hobbyist

Home grown gardens produce tastier and healthier produce than what can be purchased at a local grocery store. Most produce that is on the shelves has traveled many miles over several days to be delivered to your shopping cart. There are many other benefits to home grown produce, these benefits include improved family health, saving money on groceries, and reduced environmental impact.

Mid-sized hydroponics systems usually have roughly 20-50 plant spaces available. These systems are often set up in rows so the home grower can have either a wide variety of produce or various stages of the same produce.

Figure 6: Small Hydroponic System [3]

Many aeroponic systems are available for purchase and have a variety of plant spaces available. Figure 7 shows an aeroponic system which is designed for bigger plants. It has six plant spaces and is generally used to grow tomatoes or other large produce plants.

Figure 7: Aeroponic Home Growing System [4]

While there are some commercial aquaponic systems, many are home made. An example of an aquaponic

system is shown in figureFigure 8. Media filled beds are a simple form of aquaponics, this system consists of containers filled with a medium of expended clay, gravel, or similar. The water is then pumped over the media filled beds, while the plants grow within the medium.

Figure 8: Aquaponic Media Filled Beds [5]

Commercial

Hydroponics has recently been growing in popularity amongst commercial growers because of its efficiencies. Because of this, greenhouses that are used for commercial hydroponics produce 20-25 percent more than traditional greenhouses. With some crops, such as tomatoes, hydroponics has the potential to double yield compared to traditional greenhouses.

Growing tomatoes with hydroponics has achieved a yield as high as 650 000 lbs per acre, with more common yields being from 40 000 to 60 000 lbs per acre. Other crops such as lettuce and peppers produce four times the yield per acre. [31] Commercial farmers will be able to reduce costs on pesticides and other chemicals as well. Hydroponic greenhouses allow farmers to grow year round increasing profits.

Figure 9: Hydroponic Techniques Use Less Space [6]

One downside to a hydroponic greenhouse is the substantially higher initial investment compared with traditional agriculture. On the commercial scale initial start up costs are likely to be in the hundreds of thousands of dollars. Ohio State University has a hydroponic crop program which cultivates hydroponic vegetables. They have outlined a potential budget for a hydroponic system at different scales. For a 3072 ft^2 greenhouse plants would occupy 90% of the space. With one of these greenhouses a farmer can produce just fewer than 60 000 heads of lettuce per year and earn just over 15 000 dollars on their return. Obviously the more greenhouses there are the more return on investment that will be made.

From the commercial standpoint automation allows a farm to be operated by fewer employees, while growing more crops, maximize profits and efficiency. Table 1 contains the final calculations for a budget of four 3 072 ft^2 green houses growing hydroponic lettuce, outlined by the Ohio State University study.

	Variable Costs	Fixed Costs	Total Costs	Return over Variable Costs	Return on Investment
Per head	0.63	0.14	0.77	0.47	0.33
Per ft^2	12.12	2.61	58.91	36.16	25.73
Per house	148 928.34	32 052.34	180 980.69	111 089.66	79 037.31
Per Acre	527 939.34	113 623.06	641 562.39	393 804.16	280 181.10

Table 1: Ohio State University Cost Analysis [24]

Large scale aquaponics greenhouses can save tremendous amounts of water. Other non-recirculating systems can use five to ten thousand gallons of water per pound of production; for farmers operating with low water availability aquaponics offers a viable solution [12]. Since aquaponics is a closed loop system the

only water lost is due to evaporation and transpiration. A popular market fish such as tilapia or trout is raised along with the produce providing the farmer with two streams of income and an important protein source in developing countries. A typical commercial aquaponic installation growing lettuce can produce an average of 7 500 lbs of fish and 194 400 heads of lettuce per year using the raft method [32]. The raft configuration is very common in aquaponics, where the plants are grown on rafts that float on the water, separated from the fish. An example of this is shown in Figure 10.

Figure 10: Lettuce Grown Using Raft Aquaponics [7]

Currently there are very few commercial aquaponics systems because of lack of knowledge and a difficulty in sustaining the system without trained personnel or advanced control systems. As such hydroponics remains the leading soil-less agriculture technique used at the commercial level; however, aquaponics is gaining some traction due to its inherently organic nature.

2.3 Justifications

There are many reasons to utilize CEA on a global scale; conservation of resources and reduction in CO_2 emissions are just a few.

Conservation of Resources Arguably the most desirable feature of CEA is the efficiency allowed. Soiless CEA uses less water, less power, less nutrients and little to no pesticides or herbacides. Precise control over a growing operation enables growers to realize savings of 15-50% for energy, water, chemical and pesticide applications [28]. The goal of most farmers is to turn a profit; after the inital investment farmers are rewarded by increased yeilds and lower cost per unit resulting in higher profit margins, in addition to becoming more sustainable.

CO_2 Emission Reduction By using less water and power CEA results in a reduction in CO_2 emissions from power plants. More directly, harvesting is less resource intensive in soiless systems so less fuel is burned in farm equipment reducing CO_2 emissions further. Furthermore, by creating more farms closer to where the food is consumed, CO_2 emissions from transporting produce can be reduced.

2.4 Current Research and Projects

Internationally there are many projects promoting the use of sustainable controlled environment agriculture, below is one example.

Science Barge

The science barge is one example of a sustainable, urban hydroponic farm. This farm was designed by New York Sun Works which is a nonprofit environmental organization in New York. Their goal is to help build sustainable power and food for the city of New York as well as educate its citizens on the importance of renewable energies and sustainable food production. The farm is built on a barge in the Hudson River and is made almost completely sustainable by using numerous energy saving solutions; it also has fewer emissions and limited run off. The barge utilizes five internally regulated wind turbines, twelve solar panels with 12.5% efficiency, and a generator that runs on used vegetable oil. There is also a water recovery system that catches and stores rainwater, as well as a river water purification system. The water production system actually produces more water per day then the farm needs to operate, but this is not the case with energy. The solar and wind harvesting does not account for all of the power need which is why the generator is used. However since the generator operates with vegetable oil the farm can easily acquire some from any New York restaurant, which collectively each year supply enough waste oil to supply ten million gallons of bio-diesel fuel.

The barge grows tomatoes, cucumbers, lettuce, pepper, and herbs using a variety of hydroponic methods. A vine system, a leaf crop system, and a vertical stacking system are all used to grow the vegetables. The greenhouses are controlled by the VersiSTEP integrated control system made by Wadsworth.

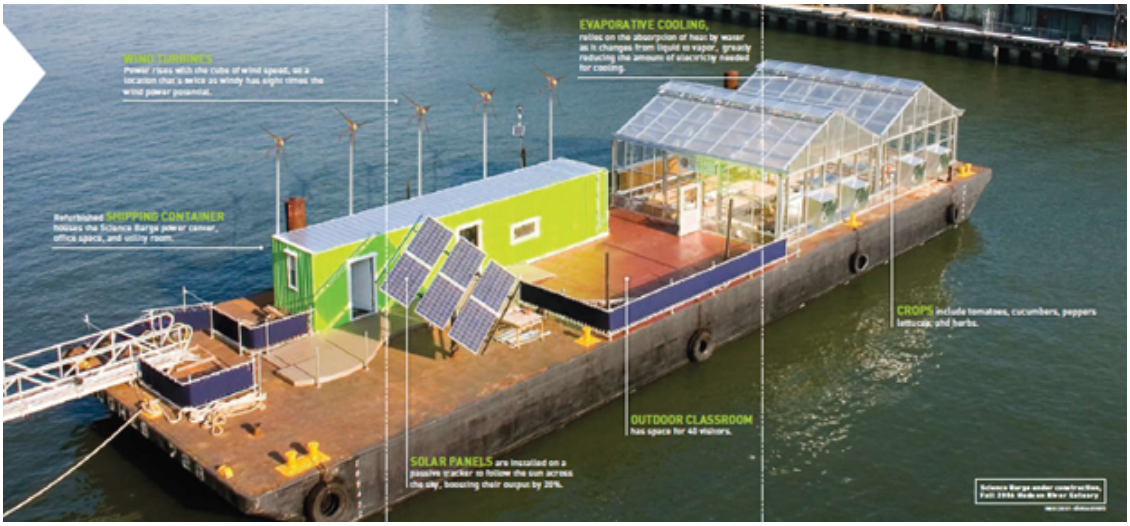

Figure 11: The Science Barge [8]

The science Barge also has a classroom on board where they host field trips for local schools to come and learn about sustainable agriculture. As mentioned before New York Sun Works is concerned with educating children about the importance of sustainable urban farming. The barge has a tour which focuses on renewable energy, ecology of the Hudson River, the water purification systems, and management of a sustainable hydroponics farm. Students also partake in a horticultural project while on board which varies in topic based on the students age and developmental level. The science barge is a perfect example of what the recent urban greenhouse movement is trying to accomplish, sustainable crops close to the people who need them and investing in education for future development [8].

2.5 Small Commercial Ventures

In addition to educational and research projects, there are several small commercial ventures developing urban farming solutions for a sustainable future.

PodPonics

PodPonics, a small company in Atlanta, has developed a portable scalable hydroponic farm system for producing locally grown produce in urban environments. They convert used shipping containers into sealed hydroponic greenhouses with artificial light. Each pod ranges from 320-450 square feet and is capable of producing more than an acre of conventionally grown crops. This is made possible by the vertical growing arrangement and tightly controlled environment. The environmental control is performed by a proprietary control system. These pods can be placed anywhere as long as there is power and water, and because each pod is self-contained you can add as many pods as demand allows. They are currently developing a 11 acre

industrial park near the airport in Atlanta into an urban farm which is planned to have production equal to a 150 acre farm.

Figure 12: Small Urban Farm [9]

The Farmery

The Farmery, a start-up in Raleigh North Carolina, is developing a different urban farm solution using shipping containers. It is a combination farm and retail space. The idea being that most of the food people buy there is picked just before paying for it, ensuring maximum freshness and a closer connection to the source of your food. Any food not sourced directly from the Farmery is sourced from small local farmers. The Farmery utilizes aquaponics and vertical gardening for higher growth per square foot. It is still in the fund raising and prototyping stages. According to the developer Ben Greene, they lacked the funding necessary to purchase an environmental controller for their current prototype. They rely on an array of timers, thermostats and humidistats to maintain the environment.

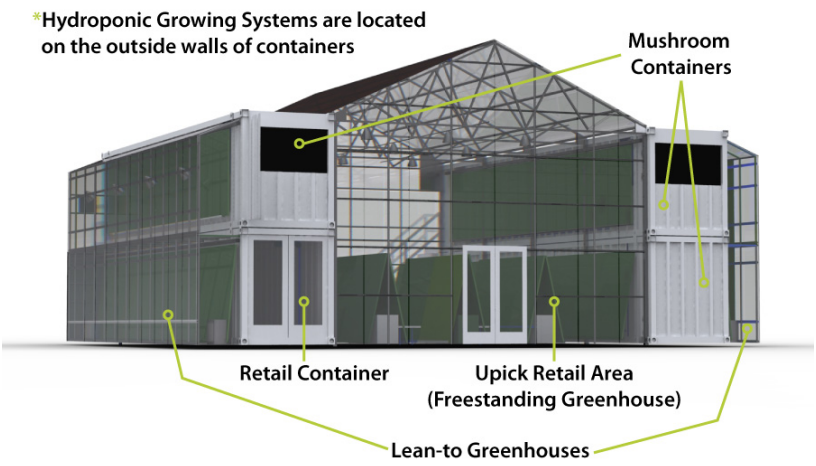

Figure 13: The Farmery [10]

2.6 Existing Products

There are a number of integrated control systems available for greenhouses; however, many are under featured or prohibitively expensive for non-commercial farmers. Often times smaller farmers resort to using an array of different solutions such as individual timers and thermostats. Unfortunately these systems are disconnected from each other and can lead to inefficiencies, sometimes even working against one another.

CAP CGC-1e

This controller is designed to operate smaller scale greenhouses. The system consists of a 24 hour timer, and four outlets which are capable of controlling temperature, humidity and lighting. Four additional outlets are controlled independently by two separate timers which can be used for hydroponic pumps or CO_2 enrichment. The limited amount of inputs and outputs impedes its ability to control larger scale operations. Additionally, it has no digital logging capabilities. Priced at $300, it is a viable solution for a small venture.

Figure 14: CAP CGC-1e Controller [11]

VeriSTEP

VersiSTEP is an integrated control system made by Wadsworth. This controller provides 24 relay outputs which are able to perform a number of operations such as controlling reversing motors for roof vents, controlling pumps and lights, as well as many other peripherals. The system features 32 analog input channels for CO_2, relative humidity, and temperature sensors as well as 8 digital detector channels to measure wind speeds and directions. There is an optional remote alarm system for alerting farmers to power failures and temperature fluctuations. While this is a full featured control system, its cost is $3 600. If expansion is desired, a 16 channel expansion module can be purchased for $1 600. This is a minor cost for a commercial grade start up but for a typical family or medium sized farm this can be a serious expense.

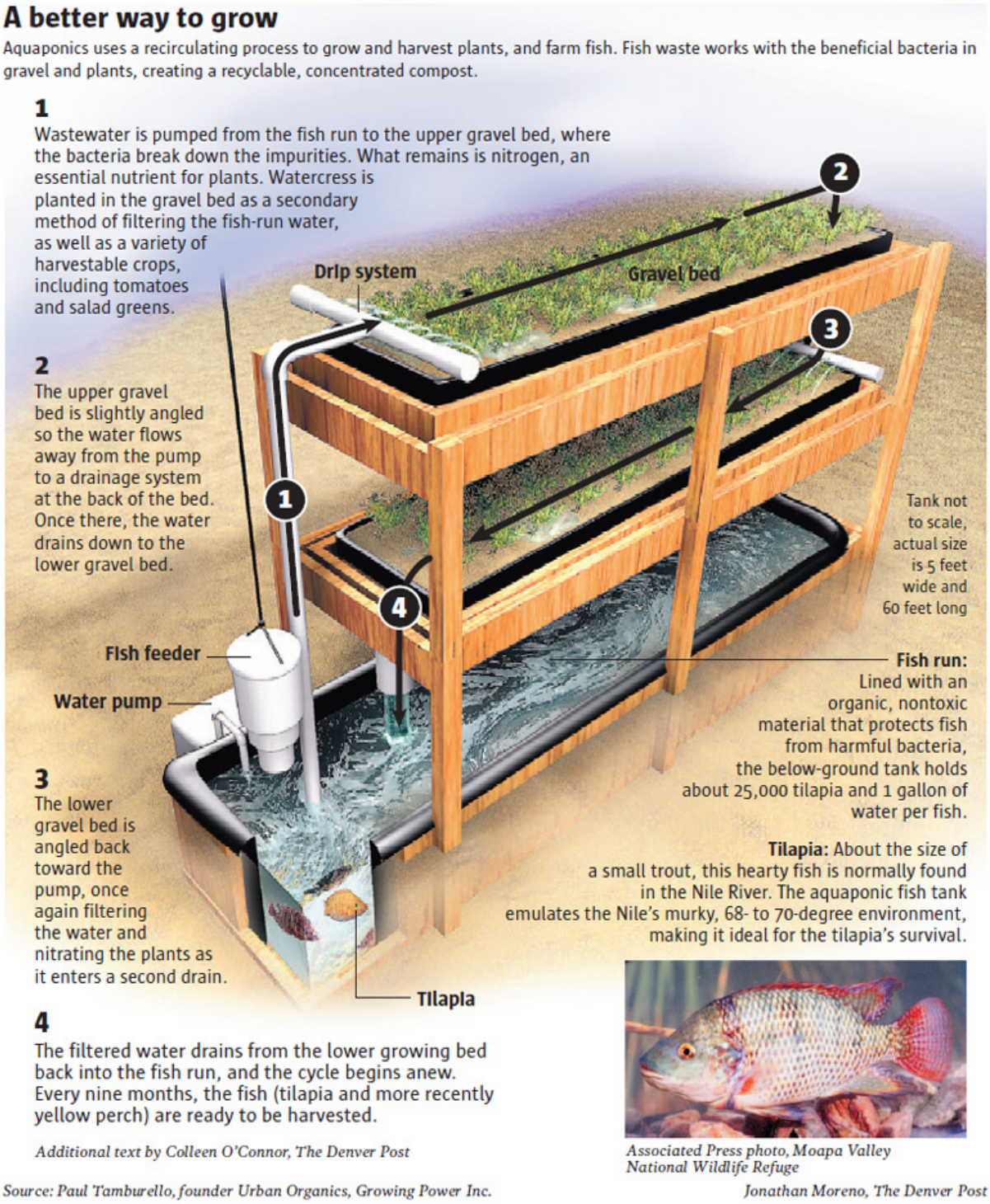

Figure 5: Overview of the Aquaponic Cycle [12]

3 Sensor Selection

Controlled environment agriculture demands fast, accurate measurements of environmental parameters such as temperature, relative humidity and CO_2. In the case of hydroponics it is also necessary to monitor nutrient water quality using pH and Total Disolved Solids (TDS) probes. The following sections will list and explain the sensing technologies available, compare, and ultimately decide on the best sensors for our application.

3.1 CO_2 Sensors

CO_2 sensors have a wide variety of uses including ventilation control, industrial incubator monitoring, food transportation, food storage, food processing, food quality monitoring, and horticulture. In horticulture CO_2 enrichment allows the space to be held at a higher temperature, safely increasing the photosynthesis rate, resulting in faster growth. According to a study conducted by Bruce Kimball, a research leader for the water conservation laboratory of the U.S. Department of Agriculture, with the doubling of CO_2 from its current level of 360ppm to 720ppm there is an average of 32% increase in plant productivity [33]. There are many types of CO_2 sensors but the most common are; Severinghaus type potentiometric sensors; infrared absorption detectors; and electrochemical sensors.

Severinghaus Potentiometric

Severinghaus CO_2 sensors don't actually measure CO_2, but rather pH. The sensor itself is a glass electrode filled with an aqueous bicarbonate solution and covered by a plastic membrane which is permeable to CO_2. When CO_2 is in an aqueous solution it forms carbonic acid, which results in a pH change in the solution. This is measured by a pH probe and then converted into a CO_2 reading that can be used.

Electrochemical

Electrochemical (EC) sensors make use of an electrochemical reaction to measure CO_2. There is a CO_2 reactive layer on top of a substrate layer along with electrodes to measure the electrical characteristics of the reactive layer. There is a heater on the other side of the substrate which is used to heat the metal oxide to the operating temperature needed to detect the gas. When the gas comes in contact with the heated metal oxide it dissociates causing a transfer of electrons which results in a change in resistance or capacitance, depending on the sensor. This change in electrical characteristics is directly proportional to the amount of CO_2 that is present. These types of sensors are much better for commercial applications because they are smaller and have greater sensitivity, however they have a limited lifetime and require a significant amount of power to operate the heater.

Non-Dispersive Infrared Detectors

Non-Dispersive Infrared Detectors (NDIR) depend on the absorption of infrared light by CO_2. They have an IR light source whose beam is split and passed through two separate gas cell. One is a reference cell

usually filled with some known gas such as Nitrogen, the other is the test cell which is where the air to be tested passes through. The sensor measures the difference in absorption of IR light which can be used to determine CO_2 concentration. Some disadvantages to NDIR sensors are their relatively high cost and large size compared to other types of CO_2 sensors. They are also subject to interference from changes in ambient light and their measurements may be skewed by CO or water vapor. However NDIR has many advantages such as long life, high accuracy, and repeatability.

We chose to use the SenseAir K30 because of its high accuracy, fast response time, moderate power consumption, and its convenient operating voltage. Furthermore the sensor contains a controller that buffers and linearizes the data from the actual sensor which can be accessed over I^2C. Table 2 contains a comparison of the CO_2 sensors we considered.

Name	Type	Range	Accuracy	Resp. Time	Voltage	Power
Figaro TGS 4161	EC	350-10 000ppm	±20% @1 000ppm	1.5m	5V	250mW
Figaro TGS 4160	EC	350-50 000ppm	±20% @1 000ppm	2m	5V	250mW
MG811	EC	350-10 000ppm	?	<60s	6V	1.2W
Alphasense IRC-A1	NDIR	0-5 000ppm	±10ppm	2-4m	2V	40mW
Cozir Ambient	NDIR	0-2 000ppm	±50ppm	1m	24V	3.5mW
Vaisala GMM112	NDIR	0-10 000ppm	±2%	1m	24V	<2W
SenseAir K30	NDIR	0-10 000ppm	±30ppm	20s	5V	200mW

Table 2: Comparison of CO_2 Sensors

3.2 Temperature Sensors

There are many types of temperature sensors but the most common are thermistors, thermocouples, and resistance temperature detectors (RTD).

Thermistors

A thermistor is a resistor made from a semiconductor whose resistance is temperature dependent. The resistivity of the thermistor decreases logarithmically as temperature increases, this relationship can be seen in Figure 15.

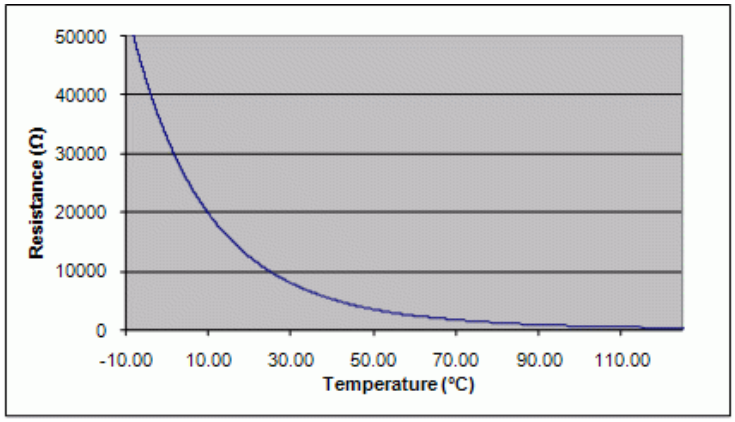

Figure 15: Thermistor Resistance vs. Temperature [13]

Thermistors are widely used because they are inexpensive, durable, and reliable within their specified operating range. A typical thermistor temperature sensor is used in the range of 0°C to 40°C, between these temperatures the resistance is most sensitive to temperature change. Implementation of thermistors is accomplished using a simple voltage divider and is measured with an ADC. The Steinhart-Hart Equation in Equation 1 is used to convert the voltage to temperature.

$$\frac{1}{T} = A + B * ln(R) + C * ln(R)^3 \tag{1}$$

where A, B, and C are manufacturing constants, and R is the thermistor resistance in Ohms

In applications where more temperature resolution is needed, a programmable gain amplifier can be used to create a minimum voltage at temperatures greater than 35°C. This allows a lower resolution ADC can be used to accomplish the same temperature resolution. An example using a programmable gain amplifier to increase resolution, provided by Microchip Technologies, is shown in Figure 16 .

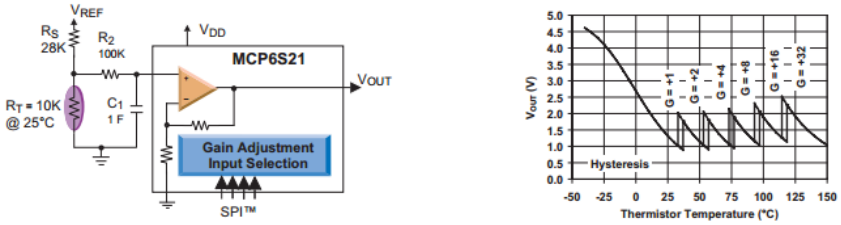

Figure 16: PGA Circuit Interfaced with Thermistor [14]

Thermocouples

Thermocouples are formed by junction of two dissimilar metals, the junction between the metals creates a potential. The potential at the junction is temperature dependent. There are several different types of metal combinations used to produce thermocouples; depending on the metal combination used they must be calibrated in different ways and have different temperature ranges. Thermocouples are dependent on a second reference junction at a known temperature, usually 0°C. The initial junction produces a temperature dependent voltage which can then be measured against the reference temperature voltage to produce a temperature reading. This conversion is done digitally because the readings tend to be nonlinear over large temperature ranges, it is represented by a polynomial equation shown in Equation 2.

$$T = \sum_{n=0}^{N} a_n \cdot v^n \tag{2}$$

There are several different types of thermocouples classified by the types of metals used. Most common are K type thermocouples which use 90% Nickel, 10% Chromium in the positive element and 95% Nickel, 2% Aluminum, 2% Manganese, and 1% Silicon in the negative element. Thermocouples are a popular choice because they are inexpensive and durable. Thermocouples require no external voltage and therefore are significantly more power efficient.

Resistance Temperature Detectors

RTDs are precision temperature dependent resistors. RTDs are made of a coiled wire of pure material such as platinum or nickel, wrapped around a ceramic core. Each metal has a specific resistance versus temperature (R vs. T) relationship which allows the change in temperature to be measured as change in resistance. This relationship is called the temperature coefficient of resistance, and is represented by α. Equation 3 shows the equation used to determine α for a specific material.

$$R_T = R_0 + R_0\alpha[T - \delta(\frac{T}{100} - 1)(\frac{T}{100}) - \beta(\frac{T}{100} - 1)(\frac{T}{100})^3] \tag{3}$$

R_T=Resistance at Temp T
R_0=Resistance at T=0°C

β and δ are determined by testing the RDT at four temperatures and solving for the resulting equations. Pure platinum has the most linear value for α at .003925 $\frac{\frac{\Omega}{\Omega}}{^\circ C}$, which is why it is the most commonly used for industrial RTDs. This linearity allows RTD measurements to be highly accurate and repeatable over a wide temperature range. RTDs are more accurate than thermocouples at a lower ranges of temperatures (-200°C to 500°C); however thermocouples can operate at up to 2 320°C. Compared to thermocouples and thermistors where readings are near instant, RTDs can take up to a few seconds to respond to changes in temperature.

Ultimately we chose to use the TI LM92 because of its high accuracy, linearized output, I^2C interface and wide supply voltage range. A comparison of temperature sensors we compared can be found in Table 3.

Name	Range	Accuracy	Voltage	Current	Interface
TI TMP275	-40-125°C	±0.5°C	2.7V-5.5V	50μA	I^2C
TI LM35	0-100°C	±0.5°C	4V-30V	56μA	Analog
TI LM92	-25-150°C	±0.33°C	2.7V-5.5V	350μA	I^2C

Table 3: Comparison of Temperature Sensors

3.3 Humidity Sensors

Relative Humidity (RH) is defined as the ratio of actual vapor pressure (P) to saturated vapor pressure (Ps). Relative humidity is temperature dependent; as temperature decreases, Ps decreases and the ratio increases resulting in a higher relative humidity for a given absolute humidity. There are several available sensor types; capacitive, resistive and thermal conductivity.

Capacitive Relative Humidity Sensors

Capacitive humidity sensors consist of a glass or ceramic substrate with a thin film of metal oxide or polymer laid between two electrodes. This dielectric absorbs or releases water based on the humidity, which changes the dielectric constant. The capacitance is dependent on the dielectric constant so a humidity change is calculated by first calculating the dielectric constant, k, using the equation shown in Equation 4.

$$k = 1 + \frac{211}{T}(P + \frac{48P_s}{T}H)10^6 \tag{4}$$

The relationship between k and relative humidity in this equation is linear. Figure 18 shows a typical response from an actual sensor. For most sensors it is linear inside of the 5-95% humidity range.

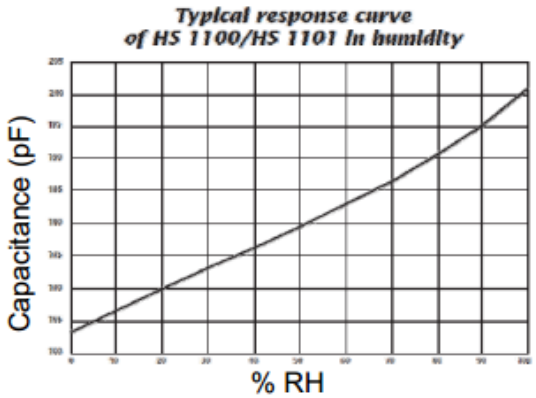

Figure 17: Relative Humidity vs. Capacitance [15]

Resistive Humidity Sensors

Resistive humidity sensors are made from noble metal or wire-wound electrodes on a substrate coated in a conductive polymer or salt. When the polymer or salt absorbs water vapor from the air ions dissociate

changing the conductivity of the sensor. This change in conductivity is measured and converted into a relative humidity reading. Resistive humidity sensors depend on AC excitation to prevent polarization of the sensor. The response time for resistive sensors ranges from 10-30 seconds and sometimes longer for large swings in humidity. The change in resistance relative to humidity is an inverse exponential function shown in Figure 18 [15].

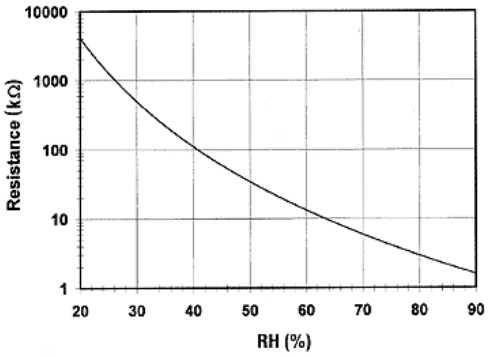

Figure 18: Relative Humidity vs. Resistance

Thermal Conductivity Humidity Sensors

These sensors consist of two NTC thermistors connected in a bridge circuit. One of the Thermistors is completely contained in dry nitrogen while the other is exposed to the air. As current gets passed through the thermistors the resistive heating increases their temperature. The heat of the sealed resistor differs from the heat of the exposed resistor because of the water vapor in the air increases the thermal conductivity compared to the dry nitrogen.

Table 4 shows the humidity sensors that we compared. Ultimately we decided to use the Honeywell HIH6131 because its output is already compensated for temperature, and the communication is I^2C which is common to all of our sensors.

Name	Repeatability	Accuracy	Voltage	Power	Output
Sensirion SHT15	±0.1%	±2%	2.4V-5.5V	$30\mu W$	Digital
Maxdetect RHT03	±1%	±2%	3.3V-6V	$5mW$	Serial
Honeywell HIH6131	±0.1%	±5%	2.7V-5.5V	$2.2mW$	I^2C

Table 4: Comparison of Humidity Sensors

3.4 pH Sensors

In aquaculture knowing the pH of your water is very important. Plants and animals depend on a particular pH to survive. pH is measured using a specially made electrode. pH probes are made with two electrodes, one as a reference and one that measures the pH of the solution. The measurement electrode is constructed with

special glass that allows for the transmission of hydrogen ions. This glass is chemically doped with lithium ions to enable the electrochemical reaction with the hydrogen ions to occur. The amount of hydrogen ions present in a solution is directly proportional to the pH; the more hydrogen the higher the pH. The electrochemical reaction produces a voltage that can be measured against the reference electrode and produce a pH level.

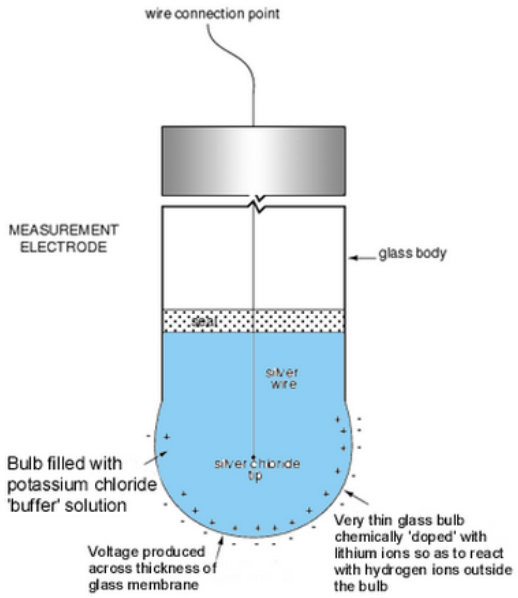

Figure 19: Measurement Electrode Illustration [16]

The reference electrode is usually made with potassium chloride solution with a pH of 7, which exchanges ions with the test liquid through a porous membrane. The job of this electrode is to produce a zero voltage so the voltage of the measurement electrode can be measured and converted into a pH level. The measurement electrode is very resistive because of the glass membrane so a amplifier with very high input impedance must be used to obtain an effective reading.

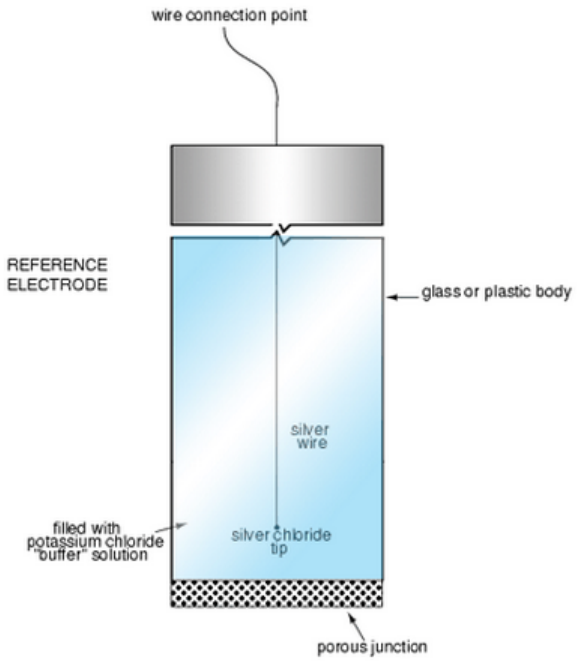

Figure 20: Reference Electrode Illustration [16]

Atlas Scientific pH Sensor This probe is manufactured by Atlas Scientific. It is a silver-silver chloride probe which can measure 0-14 pH. The electrodes are enclosed in an epoxy that Atlas Scientific claims is virtually unbreakable making the sensor very durable. It can operate from 1-99°C and has a response speed of 1 second. This sensor produces an analog output. The sensor has a 30" long cable with a BNC connector and is priced at $55.00 for a single unit.

Figure 21: Atlas Scientific pH Probe [17]

HI 1286 This pH Electrode Probe is manufactured by Hannah Instruments. It is a spherical shaped probe with polymer on the interior. It has a double junction made of PTFE and uses silver chloride as a reference solution. It can measure from 0 to 13 pH and has a BNC connection. This probe is priced at approximately $50.

3.5 Conductivity Sensors

Electroconductivity sensors are use to measure total dissolved solids (TDS) in a solution; this is useful for determining the concentration of nutrients in solution. Electroconductivity sensors use two to four platinum electrodes to measure the conductivity of a solution [34]. An AC voltage is supplied to one of the electrodes and is then measured at the other electrode. AC is used to prevent electrolysis of the solution. The voltage measured at the opposite electrode is dependent on the conductivity of the solution. Conductivity is inversely related to resistance, shown in Equation 5 [35].

$$\sigma = K_C \frac{1}{R} \tag{5}$$

The construction of the probe dictates the cell constant, K_C, but can be determined by measuring the resistance in a solution of known conductivity. Conductivity measurements are dependent on temperature because it depends on ionic motion. To compensate for this, electroconductivity sensors usually have integrated temperature sensors.

4 CAN Network

The backbone of our design is a network on which the master controller communicates with sensor and control nodes. Originally we contemplated using Ethernet but quickly realized that it would increase the cost and complexity of our network. After investigating several more communications protocols, and briefly considering designing our own, we chose to use CAN.

CAN, short for Controller Area Network (CAN), is a message based multi-master serial communications protocol. Originally developed in 1983 by BOSCH GmbH for connecting electronic control units in cars. The original CAN standard has spawned several variations developed for specific applications; most notably it has been adopted as a robust industrial automation standard. There are several reasons we chose CAN as the foundation on which we built our network of sensors and controls. CAN's low cost of implementation and high reliability made it an easy choice. The CAN standard defines the data-link layer and part of the physical layer. Error correction and bus arbitration are completely transparent simplifying program design.

Figure 22 shows a diagram of the components of our network and how they fit into the layered OSI network model.

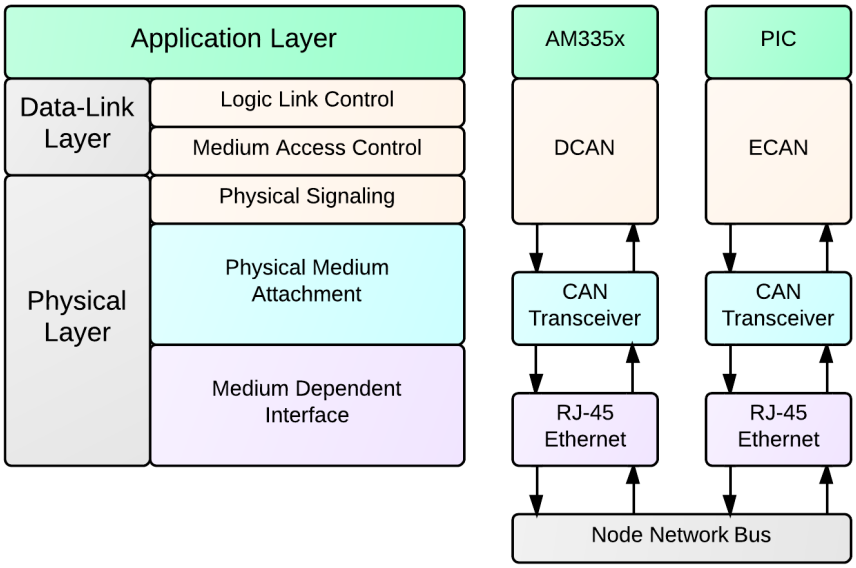

Figure 22: OSI Layered Model

4.1 CAN Specification

4.1.1 Physical Layer

The CAN specification defines many aspects of the physical layer including a bus with a nominal impedance of 120Ω and 120Ω terminating resistors at each end. The connectors, conductors, and operating voltages are left open to the designer.

CAN bus Signaling

At the signal level CAN serial data is a differential bipolar signal with two states. CAN uses inverted logic so '0' is represented by the dominant bus state and '1' is represented by a recessive bus state. The inverted input logic of the CAN bus is shown below.

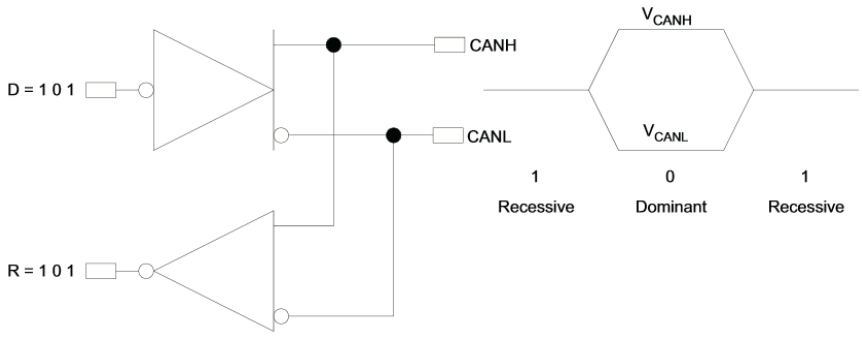

Figure 23: Inverted CAN Bus Logic [18]

Bit Encoding

CAN uses an inverted Non Return to Zero (NRZ) encoding scheme. Logic levels are held for the duration of the bit time, strings of 1's or 0's are represented by the bus remaining in a single state for the duration of the string. Because of the NRZ encoding scheme there are no edges that can be used for resynchronization during long continuous transmissions of 1's or 0's, so bit-stuffing is employed to ensure synchronization. After five bits of the same logic level a complementary bit is added to provide an edge for resynchronization. These bits are de-stuffed by receivers and the original message remains.

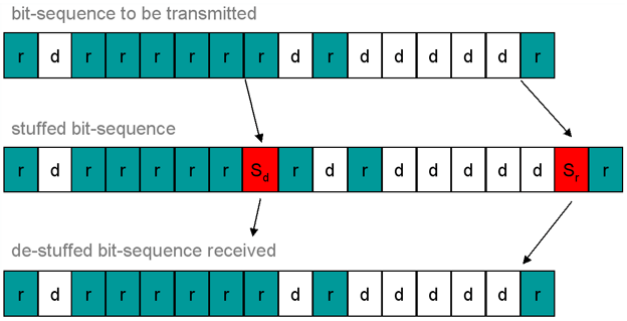

Figure 24: CAN Bit-Stuffing [19]

Bit Timing

CAN supports bit rates up to 1 Mbps and lower than 1 kbps, the Nominal Bitrate (NBR) is defined as the number of bits transmitted per second by an ideal transmitter with no resynchronization.

$$NBR = f_{bit} = \frac{1}{t_{bit}}$$

Bit time is the duration of a data bit on the bus. The nominal bit time (t_{bit}) is determined by the total length of four programmable segments of time.

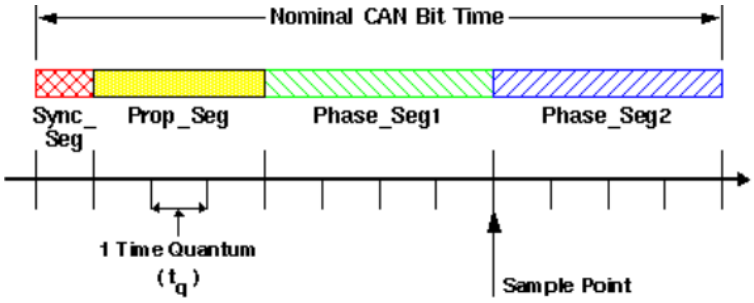

Figure 25: CAN Bit Time [20]

$$t_{bit} = t_{SyncSeg} + t_{PropSeg} + t_{PS1} + t_{PS2}$$

By adjusting the length of these segments synchronization can be achieved across nodes with different clock sources. The length of these segments is measured in Time Quanta (TQ) which is based on the system clock (F_{osc}) and the baud rate prescaler (BRP).

$$TQ = \frac{2 \times BRP}{F_{osc}}$$

The synchronization segment is the first part of the bit time, and its length is fixed at $1TQ$; this is when edges on the bus are expected to occur. The second piece is the propagation segment which is used to compensate for the physical propagation delays of the network it can be set as 1-8TQ. Propagation delay is a signal's round trip time on the network including the propagation delay of the bus cabling (t_{bus}), the output driver delay (t_{drv}) and the input comparator delay (t_{cmp}).

$$t_{prop} = 2 \times (t_{bus} + t_{cmp} + t_{drv})$$

The Sample Point is the point in time at the end of the first phase buffer segment, when the bus level has been read and interpreted as either recessive logic '1' or dominant logic '0'. The information processing time (IPT) begins at the sample point and is the amount of time needed for the logic to determine the bit level of the sampled bit. The IPT is less than or equal to 2TQ.

The two phase buffer segments on either side of the sample point are used to compensate for phase errors on the bus. These segments may be lengthened or shortened by resynchronization. The first phase buffer is

programmable from 1-8TQ. The second phase segment is equal to the sum of the first phase buffer and the IPT.

Synchronization

CAN is a synchronous serial communication standard, that is there is no separate clock signal, so nodes are dependent on the CAN frame itself for continuous resynchronization. Hard synchronization occurs on the first recessive-to-dominant edge following a period of bus inactivity. Resynchronization occurs continuously during message reception, this is accomplished through the use of a Digital Phase Lock Loop (DPLL) which compares the actual position of a recessive-to-dominant edge on the bus to the position of the expected edge within the Synchronization Segment and adjusting the bit timing as necessary.

Oscillator Tolerance

The CAN specification specifies a maximum oscillator tolerance of 0.5% for operation at speeds up to 1Mbps, with some modifications to the protocol this limit may increased to 1.58% at speeds up to 125 Kbps. The onboard oscillator of the PIC has a tolerance of up to 2% which exceeds the maximum stated by the CAN specification. In our small scale tests this had no impact on functionality but in larger networks the addition of a quartz oscillator to nodes should greatly increase reliability and performance of the network allowing higher data rates.

4.1.2 Data-Link Layer

Message Frames

The CAN specification defines four different frame types: data, remote, error and overload. Data frames contain data for transmission from one node to another. Remote frames are requests for specific node to begin transmission. Error frames are transmitted by any node detecting an error. Overload frames are used to add a delay between remote and data frames. Data frames are the only ones capable of data transmission.

Data Frame Format

The data frame consists of four sections: the arbitration field, the control field, the data and the CRC.

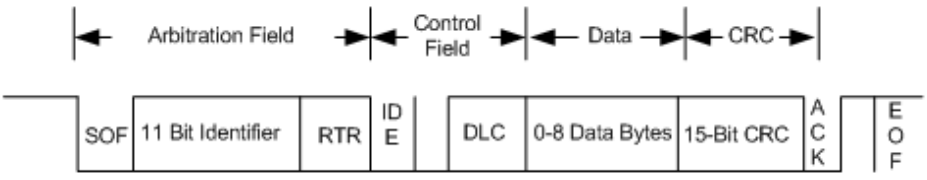

Figure 26: Standard Data Frame Format [21]

In standard CAN messages the arbitration field contains the start of frame bit, the 11-bit message identifier, and the RTR bit. The Extended CAN protocol allows the use of a 29-bit identifier; however for our network the 11-bit standard identifier is more than adequate. Table 5 shows a complete standard CAN frame.

Field name	Length (bits)	Description
Start-of-frame (SOF)	1	Denotes the start of frame transmission
Identifier	11	Identifier which also represents priority
Remote transmission request (RTR)	1	Remote for activity
Identifier extension bit (IDE)	1	Use 29 bit identifier
Reserved bit	1	Must be dominant 0
Data length code (DLC)	4	Number of bytes of data (0–8 bytes)
Data field	0 - 64	Data to be transmitted by byte
CRC	15	Cyclic redundancy check
CRC delimiter	1	Must be recessive 1
ACK slot	1	Acknowledgment bit
ACK delimiter	1	Must be recessive 1
End-of-frame (EOF)	1	Must be recessive 1

Table 5: Standard CAN Data Frame

Bus Arbitration

Because of the message based nature of CAN, bus access is event driven resulting in occasional simultaneous bus use. CAN handles bus arbitration using a non-destructive bit-wise arbitration method. Message priority is assigned in the arbitration field by the value of the identifier field, lower identifiers get higher priority. When a node is transmitting it is also listening to the bus for its own transmission; if a node is transmitting a recessive bit but detects a dominant bit it ceases transmission. If two nodes were to access the bus simultaneously whichever node holds the bus in the dominant state longer wins the arbitration and continues with message transmission as if the conflict never occurred. The message with lower priority is transmitted once the bus is free again. Data frames always win arbitration against remote frames.

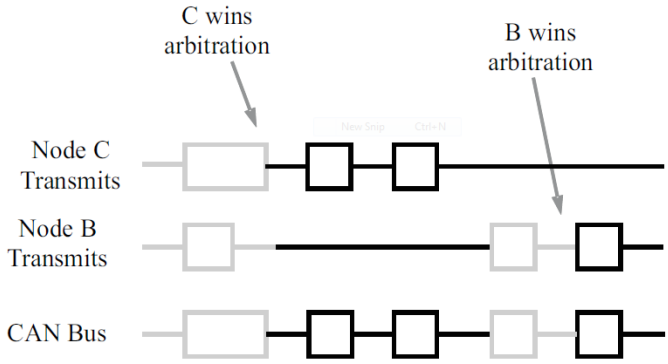

Figure 27: CAN Arbitration [18]

Error Checking

In addition to the differential signal CAN gains robustness from extensive use of error checking. The CAN protocol uses five different methods for error checking, if any one method fails an error frame is generated by the receiving node forcing the transmitting node to resend the message until it is properly received.

Error checking at the message level is handled by the CRC and ACK fields of the data frame. The 16-bit CRC field is a 15-bit checksum of the preceding data field with a 1-bit delimiter. The ACK field consists of an acknowledge bit and an acknowledge delimiter bit. A form check is also applied at the message level checking that the SOF, EOF, ACK delimiter and CRC delimiter bits are all recessive. At the bit level transmitting nodes check the bus level against the output state, if they do not match and the node is transmitting data an error is generated. The final method for error checking is the bit-stuffing rule, if five consecutive bits are the same and the sixth is not the complement an error is generated.

4.2 Hardware

4.2.1 Cabling and Interconnects

We wanted our network to be implemented using a single cable carrying both data and power for sensors. The CAN standard calls for a twisted pair cable with a nominal cable impedance of 120Ω for data transmission and an additional pair for a common ground and bus power. The CAN specification calls for no particular connections but several connectors have been standardized. We chose to use the RJ-45 connector for our interconnects because of its standardized pin-out allowing us to use readily available Ethernet cables for testing and development. Ethernet cables have a nominal impedance of 100Ω which is lower than the 120Ω specified by the CAN standard. This did not affect the CAN networks performance in our small scale tests.

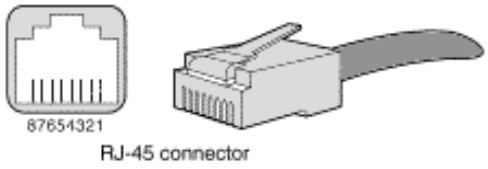

Figure 28: RJ-45 Pinout [22]

4.2.2 Bus Power

Our nodes depend on 5V to operate. The current rating for 24-guage Cat 5e cabling is 577mA per conductor, using two conductor for the transmission of bus power allows for a maximum bus current of 1.154A. With a operating voltage of 5V the bus is capable of transmitting up to 5.77W of power and is able to operate up to six nodes if they draw roughly 200mA or 1W each. If you take the voltage drop in the cabling of the bus into account, using cables of any appreciable length the nodes soon begin to malfunction.

Pin #	Signal	Description
1	CAN_H	CAN High Signal
2	CAN_L	CAN Low Signal
3	CAN_GND	Ground
4	CAN_RST	Node Reset Signal
5	CAN_GND	Ground
6	CAN_V+	Bus Power
7	CAN_GND	Ground
8	CAN_V+	Bus Power

Table 6: RJ45 Network Pinout

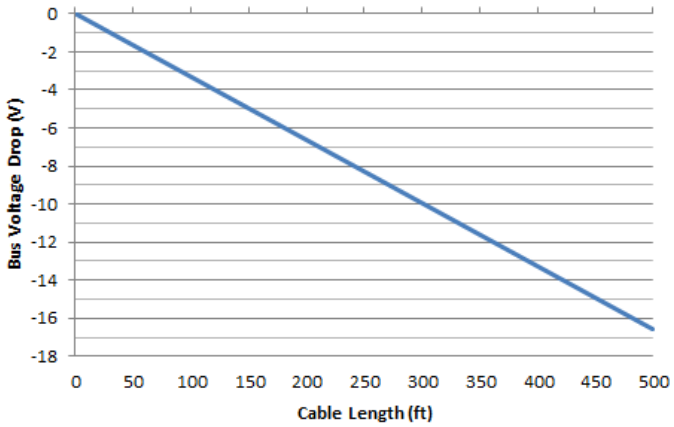

Figure 29: Bus Voltage Drop at Bus Current 1.154A

To overcome the voltage drop limitations we increased the bus voltage. A table showing different bus voltages and the number of nodes that can be run on the network can be found in Table 7.

Bus Voltage	Bus Power	Nodes
5V	5.77W	6
12V	13.85W	14
24V	27.7W	28
36V	41.54W	42
48V	55.39W	55

Table 7: Bus Voltage vs. Number of Nodes

Nodes still need a stable 5V supply to operate so a voltage regulator was implemented. The voltage regulator must be capable of operation from a wide range of voltages to handle the large voltage drops experienced with long cable lengths. The node voltage regulator must be supplied with a minimum of 6.5V in order to function. This minimum voltage is shown by the red dashed line in Figure 30.

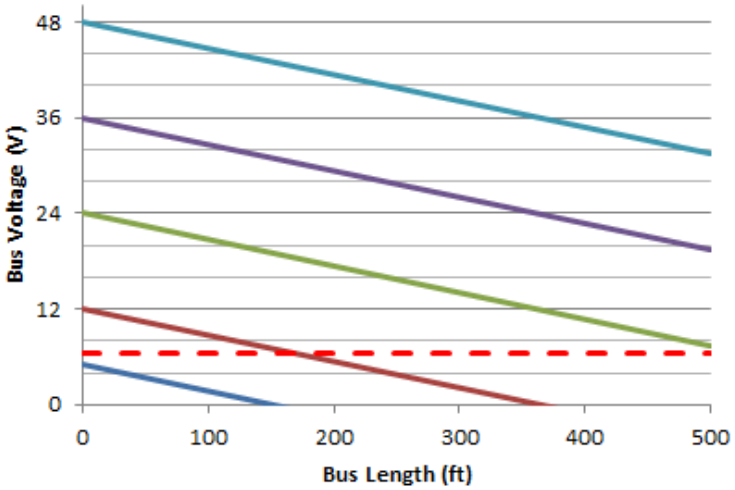

Figure 30: Bus Voltage With Different Supplies at Bus Current 1.154A

For prototyping we chose to go with a bus voltage of 24V; this theoretically allows cables of up to 500ft to be used effectively and allows us to run up to 28 nodes. Additionally we chose 24V over 36V or 48V because the power supplies were more economical and our buck converter limits the max bus voltage to 40V. We wanted to find a power supply which would supply 1A at 24V but also provide 5V for the BB to operate. We picked the CUI VGD-60-D524 to act as our main power supply. The VGD-60-D524 provides 1.5A at 24V and 4.8A at 5V making it more than adequate for our purposes.

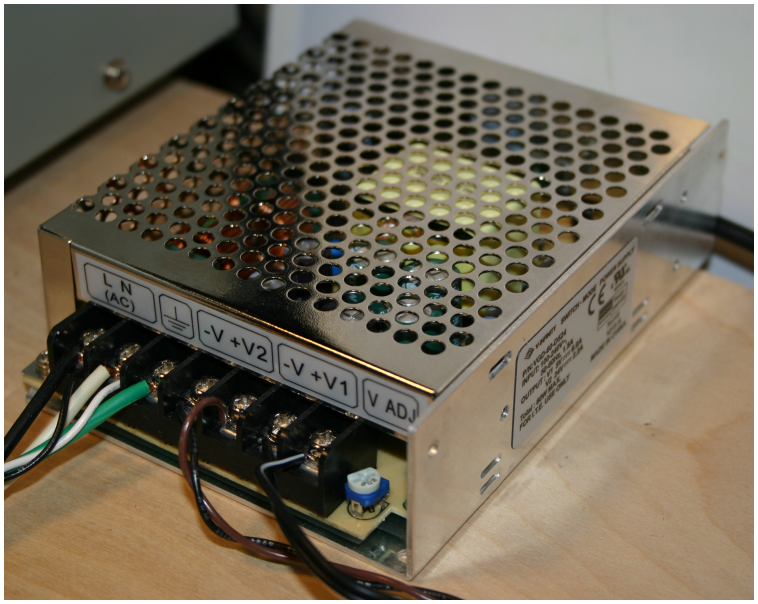

Figure 31: System Power Supply

4.2.3 Node Power Regulator

There are two main types of voltage regulators, linear and switching. Linear regulators work by acting as a voltage divider using the resistance of a transistor to dissipate excess power as heat. The power dissipated in the linear regulator is a function of the load current; and the input and output voltages. With a small load linear regulators offer an effective method to regulate voltage but with increasing voltage differentials and load currents power dissipation in the regulator soon becomes an issue. The power consumption calculations for a node using a linear regulator with a node current of 200mA and a bus voltage of 24V are shown below.

$$P_{Dissapated} = (V_{in} - V_{out}) \times I_{Load} = (24V - 5V) \times 200mA = 3.8W$$

$$P_{Dissapated} = V_{Load} \times I_{Load} = 5V \times 200mA = 1W$$

$$P_{Node} = P_{Dissapated} + P_{Load} = 3.8W + 1W = 4.8W$$

With a bus power of 27.7W we are limited to six nodes, other than allowing nodes to operate with the voltage drop in the cabling this solution is no better than our original 5V bus solution. Additionally for the linear regulators to operate properly when dissipating 3.8W large heat sinks would have to be added increasing production costs and weight. The heat dissipated by the regulator would also cause erroneous sensor readings necessitating insulation further increasing costs.

Switching regulators work by quickly switching a transistor on and off delivering power to an energy storage element such as an inductor. Because most of the energy lost in a switching regulator is due to switching losses and not conduction losses they often have efficiencies greater than 90%. For our design we chose to use a buck converter; buck converters are a type of switching regulator which steps a higher volt down to a lower voltage. We chose the Texas Instruments LM2671-5.0 because of its high efficiency and wide input voltage range of 6.5-40V. The LM2671 also features an ON/OFF pin than when pulled low shuts down the buck converter. We connected the ON/OFF pin on all of the nodes to a conductor on the bus allowing for a hardware reset of all of the nodes from the BB.

4.2.4 CAN Transceiver

While both the BB and the PIC feature integrated CAN controllers they still require an external transceiver to translate the 0-3.3V CAN signals to the differential signal specified by the CAN standard and vise versa. The CAN transceiver we chose is the TI SN65HVD256 (SN65). The SN65 transceiver features short propagation delay times and fast loop times which enhances timing margins and improves performance in long and highly-loaded networks. While the transceiver itself requires 5V for operation it features an input for setting the RX logic levels allowing for compatibility with the 3.3V logic of the BeagleBone. The transceiver is also designed with many protection features to improve network robustness. We used the same CAN transceiver for all nodes in order to affirm system compatibility.

4.3 Communication

CAN messages are transmitted in frames which consist of 16 bit words. For consistency a CAN frame template was created for node communication. The first word is always the address of the node that is sending the message, the second word is always the type of message, and the remaining words of the message vary. Table 8 contains a list of recognized CAN frames. Figure 32 depicts a typical conversation on the CAN network between two nodes in the same zone. Sequence one shows the start up message that the PIC sends when it is initialized. Sequence two shows a full sequence of gathering data and adjusting control outputs.

Type	Description	Word 1	Word 2	Word 3	Word 4	Word 5	Word 6
1	Sensor Output	Address	1	Temperature	Humidity	CO_2	-
2	Control Node Confirmation	Address	2	PLUG1	PLUG2	PLUG3	PLUG4
3	Set Control Node State	Address	3	PLUG1	PLUG2	PLUG3	PLUG4
255	Start-up Message	Address	255	-	-	-	-

Table 8: PIC Can Frames

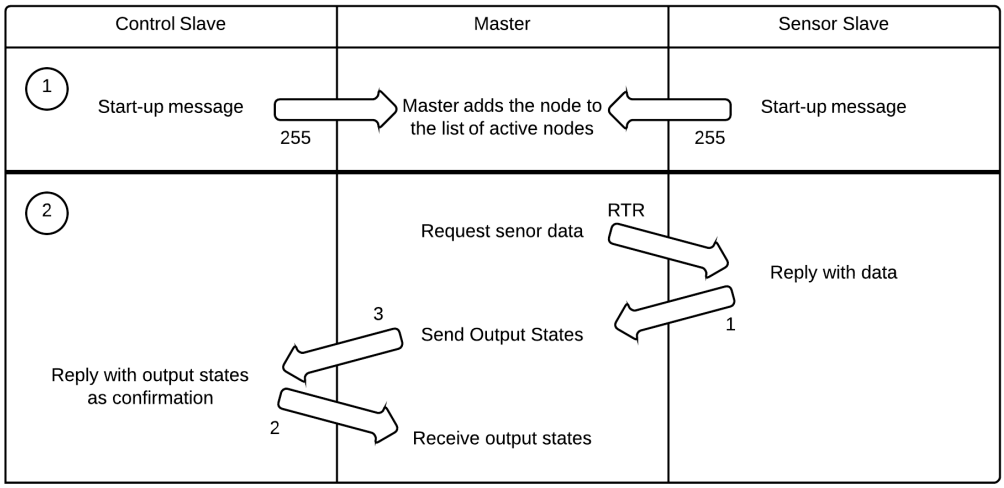

Figure 32: CAN Message Sequence

5 BeagleBone

The BeagleBone is a low cost development board for the TI Sitara AM335x ARM Cortex A-8 processor. The hardware is open source and is produced by Texas Instruments in association with Digi-Key. It runs at 720MHz and has 256MB of RAM. It has the ability to run any flavor of Linux that can run on an ARM processor; this includes but is not limited to Angstrom, Ubuntu, and Android. We decided on using the BeagleBone for its small size, open-source hardware schematics, built-in LCD controller, integrated Ethernet, integrated ECAN module, and abundance of general purpose input-output (GPIO) pins.

The BeagleBone has two 2x24 headers on a credit card sized board which are muxed to accommodate the many functions of the AM335x. These GPIO pins can assume a wide variety of functions including a 24-bit LCD controller, two CAN interfaces, two I^2C interfaces, and an abundance of ADCs.

The BeagleBone ships with Angstrom Linux, a distribution geared towards small embedded computers. This image includes everything necessary to start the BeagleBone, including a web server that guides you through initial setup. Using this image we verified that the BeagleBone was in working condition. We used Minicom, a Linux TTY emulator, to communicate with the BB through the USB-Serial interface. Once fully started and connected to the internet it is also possible to communicate with the BeagleBone over SSH.

Some alternatives to the BB would have been the RaspberryPi, Gumstix, or PandaBoard. All of these are similar compact, low cost embedded Linux computers. We chose the BB over these because of its availability and long term support from TI, as well as its growing user community. Throughout the remainder of this document the BeagleBone will be referred to as BB.

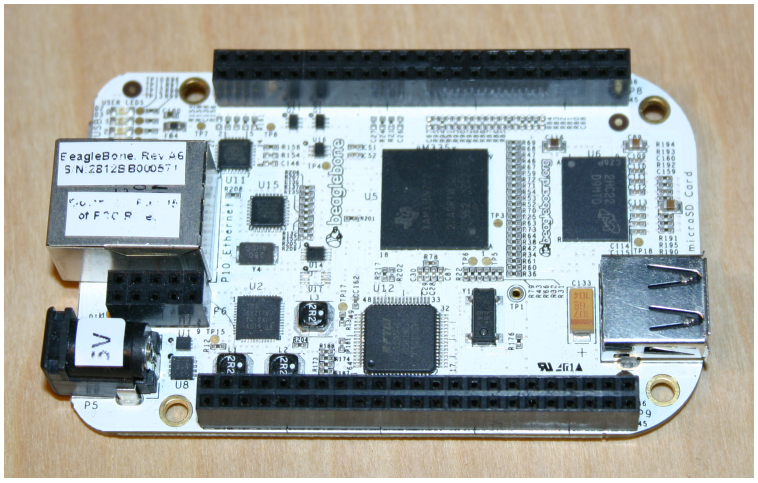

Figure 33: Our BeagleBone

Optional Capes

Available for the BB are 'capes' that plug into the GPIO expansion headers and allow the BB to assume extra functions. Each cape has an identifying EEPROM; the BB communicates with the EEPROM over

I^2C to identify the attached cape. When the BB recognizes a new cape it makes changes to the pin muxing on the expansion header as well as executes any necessary initialization steps for the cape to function. Some available capes include LCD capes in varying sizes, a battery power cape, and a CAN cape. While this is convenient and would have made development much faster, the capes were expensive and didn't have appropriate feature sets. Instead we opted to design our own cape which includes CAN, I^2C, and a 4.3" LCD touchscreen.

In order to initialize these peripherals without plugging in a cape EEPROM one must make modifications to the Angstrom kernel. Alternatively an EEPROM could be programmed with the appropriate data listed in Table 9. Because we are not using any other capes on our BB we decided to force the initializations of these peripherals directly; however, in the future an EEPROM would make kernel maintenance easier and allow our cape to be used easily by others. The following sections describe briefly how the BB operates, and how the hardware and software was modified to accommodate our peripherals.

Name	Offset	Size (bytes)	Contents	Format
Header	0	4	0xAA, 0x55, 0x33, 0xEE	
EEPROM Revision	4	2	Revision number of the overall format of this EEPROM = A1	ASCII
Board Name	6	32	Name of board so user can read it when the EEPROM is dumped	ASCII
Version	38	4	Hardware version code for the cape	ASCII
Manufacturer	42	16	Name of the manufacturer of the cape	ASCII
Number of Pins	74	2	Number of pins used by the cape including power pins	Decimal
Serial Number	76	12	Board serial number: WWYY&&&&nnnn, WW = 2 digit week of year, YY = 2 digit year of production, &&&& = Manufacturer code, nnnn = Incrementing weekly production number	
Pin Usage	88	148	Two bytes for each of the 74 expansion header pins for muxing	HEX
DD_3V3EXP Current	236	2	Maximum current in milliamps	HEX
VDD_5V Current	238	2	Maximum current in milliamps	HEX
SYS_5V Current	240	2	Maximum current in milliamps	HEX
DC Supplied	242	2	Indicates whether or not the board is supplying voltage on VDD_5V, either 0 or current supplied in milliamps	HEX
Available	244	32543	Available space for cape manufacturer to store non-volatile data	

Table 9: Cape EEPROM Data

Operating System

The default operating system that ships with the BB is Angstrom. Angstrom is a small Linux distribution designed for embedded Linux microcontrollers such as the TI AM335x. We decided to use this distribution because there is well formed community support, and native support for the BeagleBone hardware.

The other possible option was to use Ubuntu. While Ubuntu is a well designed desktop operating system, it lacks the support for the BB hardware that Angstrom readily provides.

Boot Sequence

The AM335x can boot from many types of media. On the BB it boots from the MicroSD card. When booting from the MicroSD card the board looks for an active primary partition formatted in FAT32. Once this partition has been found it looks for MLO in the root folder. From here the boot sequence is a three step process.

1. Execute MLO
2. MLO finds and executes the bootloader, U-Boot
3. U-boot finds and executes the Linux kernel

MLO MLO is a signed version of X-Loader, a stage one boot loader. By default it searches for a file named u-boot.bin on the same partition as itself, loads it into SDRAM, and executes it.

U-Boot U-Boot is a second stage boot loader which sets the kernel boot arguments, and then passes control of the system to the kernel. By default is looks for the kernel image named uImage. Boot arguments can be passed to the kernel by U-Boot by placing a file named uEnv.txt in the same location as U-Boot, alternatively one can issue commands directly to U-Boot as it starts if the user is connected over TTY.

5.1 Modifying the Kernel

In order for us to use the CAN and LCD modules we must first modify the kernel to support and initialize them. This section first gives the reader introductory knowledge about GNU/Linux and the steps to prepare a build environment for the kernel. Lastly we describe the process by which we modified the Angstrom kernel to support our devices.

5.1.1 A Brief Overview of the Linux kernel

One could divide the GNU/Linux operating system in to two levels.

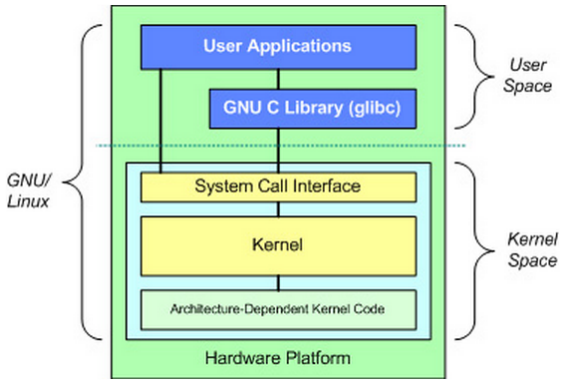

Figure 34: Fundamental architecture of the GNU/Linux Operating System [23]

On top is the user, or application space; this is where the user executes applications. On the bottom is the kernel space, where the Linux kernel exists. The GNU C library provides the system call interface, a mechanism to transition between user space and kernel space.

The kernel is further subdivided into two separate pieces. The larger piece on top which is architecture-independent, meaning it doesn't change based on the processor architecture. A majority of the source code in the Linux kernel exists as device drivers that make a particular piece of hardware usable. Device drivers form a bridge between user space and the hardware [23]. To use a particular driver it must be loaded into the kernel. This can be done while the kernel is running using *insmod*, or it can be incorporated into the kernel at compile time using *menuconfig*. Device driver code can be found under ./linux/drivers in the Linux source tree.

The smaller portion on the bottom is architecture-dependent kernel code, this is commonly referred to as a Board Support Package (BSP). A BSP also contains code specific to a particular set of hardware and is responsible for initializing board specific drivers.

The board support code for the AM335x can be found under ./linux/arch/arm/mach-omap2 in the Linux source tree. This BSP supports many mach-omap2 devices, support for the AM335x specifically can be found in board-am335xevm.c file. This file is responsible for initializing all the peripherals on the BB. From here forward board-am335xevm.c will be referred to as the board initialization file.

5.1.2 Preparing a Build Environment

The first step in making modifications to the Angstrom kernel was to set up a build environment. Angstrom offers a set of scripts which automate the setup of OpenEmbedded (OE).

OpenEmbedded is a software framework used to create Linux distributions, specifically aimed at embedded devices. At the core of OE is Bitbake, a tool similar to GNU Make, which is specifically focused on cross-compiling software for embedded Linux platforms. Bitbake recipes are similar to Makefiles and tell Bitbake which packages to build and where to find their dependencies. OpenEmbedded manages the Bitbake recipes for over 7 distributions and 5 hardware configurations. As one may imagine the package is therefore

massive and has many dependencies. Because of its size and complexity a significant amount of time was spent learning how to use it. Once mastered it is an integral tool for creating custom kernels and root file systems.

Next the appropriate modifications to the kernel are made. Finally the kernel is compiled using OE/Bitbake.

5.1.3 DCAN

The AM335x has two built-in Bosch DCAN controllers that support both CAN 2.0 A and B protocols. Using *menuconfig* the drivers were loaded into the kernel. After loading the drivers, the kernel must be forced to use the driver by adding d_can_init to the board initialization file.

Before the CAN interface can be used from the BB it must first be configured in userspace. The baudrate is set and the interface is brought up using the *iproute2* software package. Sending and receiving messages on the CAN network from userspace is accomplished using *canutils*, a set of programs that utilize the SocketCAN library. The SocketCAN library provides socket style interaction with CAN interfaces. Two programs are used: *cansend* is used to send messages to the network; *candump* displays all messages that are broadcast on the network. The output of candump can be filtered to receive messages on one address, or a subnet of addresses.

5.1.4 LCD and Touchscreen

The AM335x has a built-in LCD controller (LCDC) which can function in either raster or LIDD mode. In raster mode the LCDC constantly scans across the screen and writes the pixel data to the LCD. In the smarter LIDD mode the LCDC sends a message to the LCD telling it what to display and does not send new data until the buffer changes.

To enable the touchscreen we loaded the driver and forced the initialization by adding bone_tsc_init to the board initialization file.

In order for the LCDC to become active in user space the kernel must be configured with an appropriate driver. We modified a version of the generic DA8xx raster mode LCD driver. We created a new da8xx_panel struct which contains the timing settings for a specific LCD. See Table 11 for the settings used for our NHD display.

The driver was loaded into the kernel using *menuconfig* and the initialization was forced by adding the lcdc_init method to the board initialization file. The lcdc_init method takes a da8xx_lcdc_platform_data struct which requires an lcd_ctrl_config struct so both must be created for our LCD. Table 10 shows the settings for lcd_ctrl_config. The da8xx_lcdc_platform_data is comprised of a name, the lcd_ctrl_config, and a type name.

Key	Value
AC Bias	255
AC Bias Interrupt	0
DMA Burst Size	16
Bits Per Pixel	24
FDD	0x80
TFT Alt Mode	0
STN 565 Mode	0
Mono 8 Bit Mode	0
Invert Line Clock	1
Invert Frame Clock	1
Sync Edge	0
Sync Control	1
Raster Order	0

Table 10: lcd_ctrl_config Settings

Modifying the LCD driver

Implementing a custom LCD requires several modifications to the DA8xx driver. First a new da8xx_panel struct must be defined. These settings are specific to an LCD. See Table 11 for the settings used for the NHD display.

Key	Value
Name	NHD-4.3-ATXL#-T
Width	480
Height	272
Horizontal Front Porch	2
Horizontal Back Porch	2
Horizontal Sync	41
Vertical Front Porch	2
Vertical Back Porch	2
Vertical Sync	10
Pixel Clock	9MHz
Invert Pixel Clock	0

Table 11: Settings for 4.3' NHD Display

When properly configured the kernel shows a framebuffer in the device list. To write images to the LCD you simply copy an image to the framebuffer and the controller takes that image and pushes it to the LCD.

5.2 Database

During normal operation the system is collecting and storing three readings from every sensor node on the network every 10 seconds. That's over 25 000 data points for each sensor node every day. In addition to sensor

readings the database also contains information and settings about zones and nodes. Most importantly, all of this data must be accessible to a variety of applications. In order to store this data the system employs a relational database management system (RDBMS).

A relational database consists of one or more schemas, or collections of tables. A table is a set of data organized by columns and rows. Columns, known as fields, are identified by their name and a data type; rows, known as records, represent a single structured data item in a table. The tables are linked together via primary keys. A primary key is a field that uniquely identifies a particular record, for instance your social security number identifies you in the social security database. By separating data into multiple related tables one can eliminate redundancy thereby reducing physical storage requirements and increasing query speed. Relational databases typically implement SQL for control.

Structured query language (SQL) is a special purpose programming language designed for managing data in an RDBMS. It allows the user to alter tables; join tables; select, insert, update, and delete records. There are many open source RDBMSs that employ SQL such as MySQL, PostgreSQL and SQLite. We chose to use MySQL because it is reliably used by many high profile, large scale web applications. In addition there is a plethora of freely available programs and libraries to access MySQL databases.

A Python package, python-MySQLdb, was used to interface with the database through Python. This package allows the use of standard SQL syntax from Python. A wrapper was written around MySQLdb to provide simple calls for complex queries; this wrapper is commonly referred to as a object relational mapper (ORM).

Design

The design of our database attempts to maximize the benefits of a relational database using standard normalization techniques. There are eight tables in our database: data, data_types, nodes, node_types, output_settings, plug_types, zones, and zone_settings.

A relational diagram of the database can be seen in Figure 35.

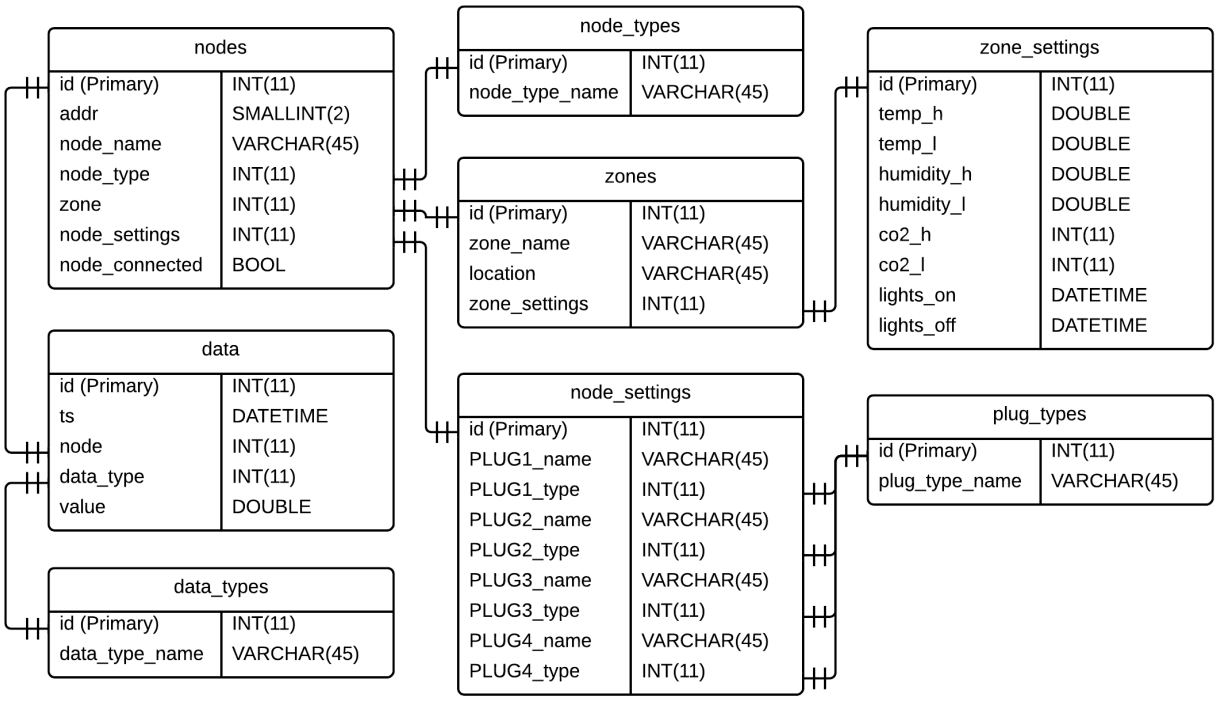

Figure 35: Relational Diagram of MySQL Database

The nodes and zones tables represent physical nodes and zones. The data table is responsible for storing all data from all sensors on the network. The other five tables are relational tables that allow new types and settings to be added without altering the other tables. For instance, if there were to be a new sensor type such as pH, a new record could be added to *data_types*. This structure allows queries to pull only as much information as they need, for instance if we want to know which zone ID a node is in we can query the *node* for the *zone*. However if we want to know the name of the zone that a node is in we can join *zones* with *nodes* and query the resulting table for *zone_name*. More complex queries that involve multiple joins are also possible, for instance determining what the temp_h is for a particular record of data.

5.3 Software

The software on the BB consists of two main parts: the daemon which handles gathering and storing sensor data and controlling zone climate; and a web interface which allows the user to modify climate set points and view climate data. This is accomplished using a combination of Python, BASH, and SQL. We chose Python as our primary programming language because it is light-weight, versatile, object oriented, and there are many packages available for a wide variety of tasks. BASH was used to create an application monitor as well as for any redundant tasks such as application initialization. A software block diagram can be found in Figure 36. The software directory listing can be found in Listing 1.

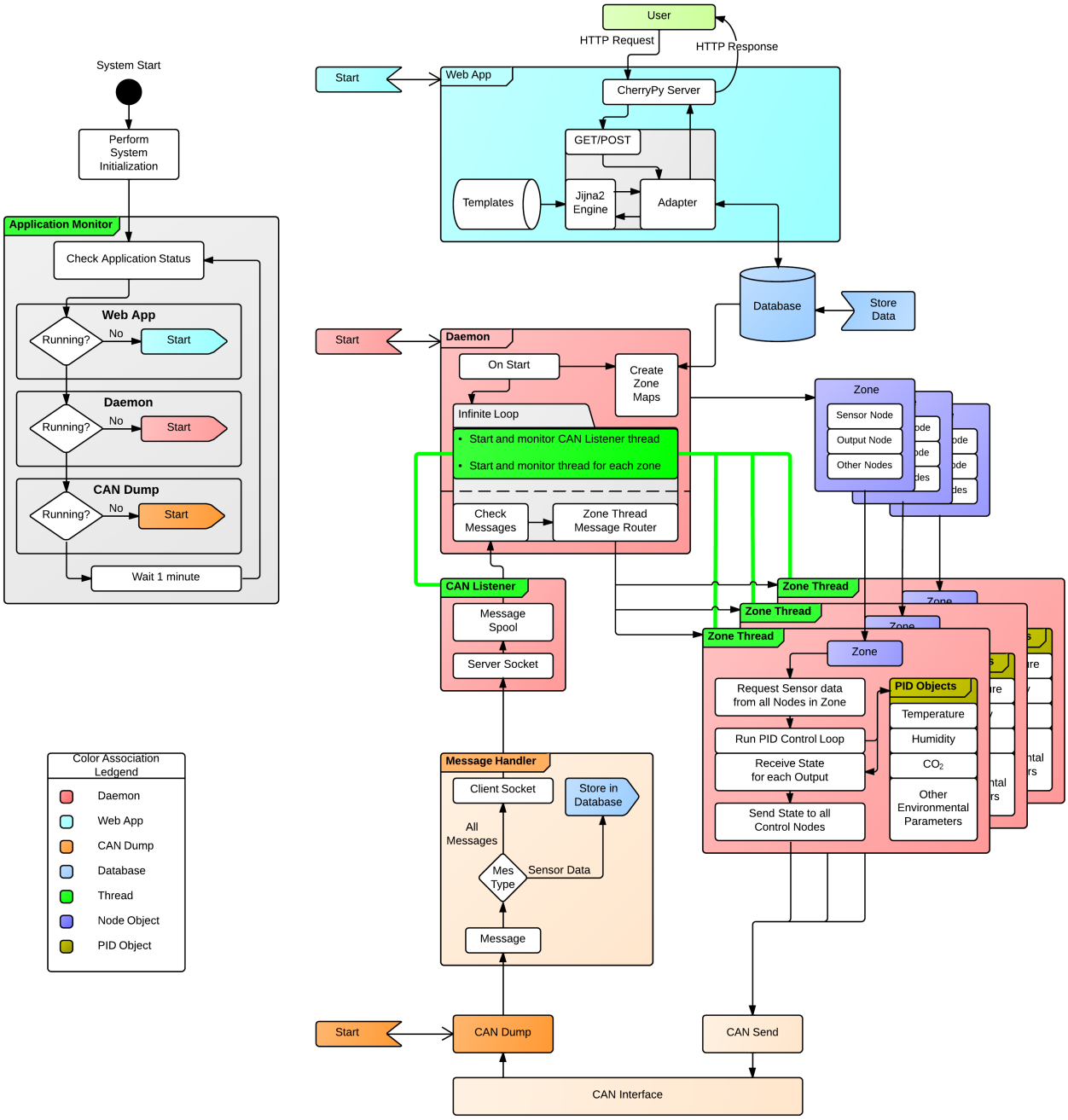

Figure 36: Software Block Diagram

5.3.1 Daemon

The daemon is the main system application. It is responsible for keeping all its child threads alive and routing messages from the CAN listener to the appropriate thread. When the application starts it queries the MySQL database for information about zones and nodes, it then creates an object representation of the nodes and zones connected to the system. It does this in order to reduce the number of database transactions and increase the operating speed of the zone threads. The application then spawns the CAN listener and a zone thread for each zone.

CAN Communication

The SocketCAN library is available for Python in version 3.3.0, but because we are using Python 2.7.3 SocketCAN is unavailable. For this reason we chose to modify *candump* to invoke a Python script, known as the handler. The handler first checks the message type; if the message type is a node start message the handler adds it to the list of connected nodes. If the message type is sensor node data the handler performs any necessary conversions and stores the data in the MySQL database. It simultaneously forwards a copy of the converted data to the CAN listener via socket. The CAN listener keeps a socket open and reads the socket for information. If a message has been received on the socket, the thread stores that information into a list. When the daemon requests information from the listener it returns and clears the list.

To send CAN messages Python invokes the *cansend* utility.

Zone Thread

Each zone thread is responsible for requesting information from the sensor nodes in the zone, executing PID loops on the sensor data, and assigning a state to plugs on the control nodes. Each zone thread is associated with one zone. Each zone can have multiple nodes. When the zone thread is started it creates a PID object for every parameter being controlled in the zone. It then sends a request to each sensor node in the zone and then waits for a response. If it does not receive a response from a particular node it will timeout; after successive timeouts the node will be removed from the list of active nodes and the administrator will be alerted. When data is received from a sensor node the PID control is executed and the output is sent to the appropriate control node in the zone; the thread then waits for acknowledgment from the node.

Each instance of a PID object is owned by a zone thread and initialized with a set point, and the three PID tuning constants, Kp, Ki, and Kd. By default Kp is set to one, and Ki and Kd are set to zero. The PID object is responsible for storing the previous error, the integral term, and the time since the last execution of the loop. Every time the PID object is executed it calculates the Δt since the last iteration, then calculates and returns an output.

A flow diagram of the zone thread can be found in Figure 37.

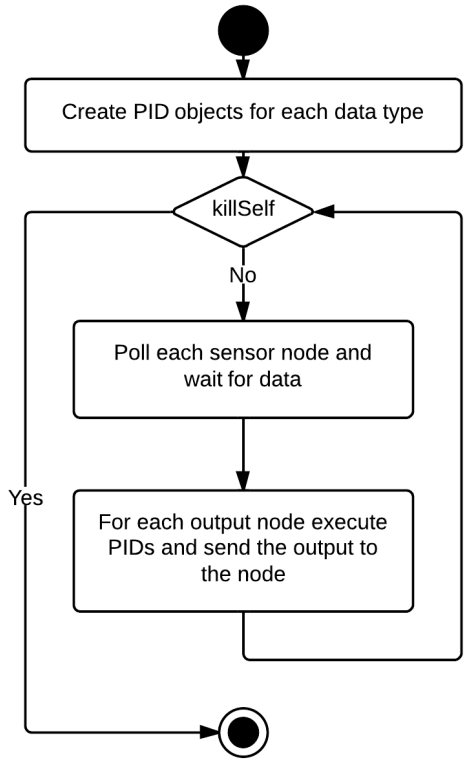

Figure 37: Flow Diagram of Zone Thread

5.3.2 Web Interface

The web interface allows the user to view and modify all the parameters of the system, for example; allowing the user to assign nodes to zones, change zone environmental parameter limits, adjust lighting cycles, and link specific environmental parameters to specific outputs on the network. The interface also allows the user to view graphs of the environmental parameters over time for particular zones. Care was taken to design the web page to be viewable on both standard computer monitors as well as smaller screens associated with smart phones and tablets.

The interface follows a Model-View-Adapter (MVA) architecture where the model and view have no knowledge of each other, instead their communication is mediated through the adapter. This differs from the standard Model-View-Controller (MVC) architecture where the view has some knowledge of the model. Often times web pages such as this are written in a combination of HTML and PHP-MySQL and are served by a web server such as Apache or LightHTTPD. Alternatively Python can be used to do all of these tasks using libraries such as CherryPy, Jinja2, and MySQLdb. CherryPy is a Python web framework which acts at a web server. Jinja2 is a Python template engine which renders the data coming from the adapter with static template files to create the view. When a GET request is received by the CherryPy server the adapter queries the appropriate data from the MySQL database and converts the data to a set of Python dictionaries and

lists which are forwarded to the templating engine for rendering. Once the view is rendered the CherryPy server sends the view to the user over HTTP. A block diagram of the web interface can be found in Figure 38.

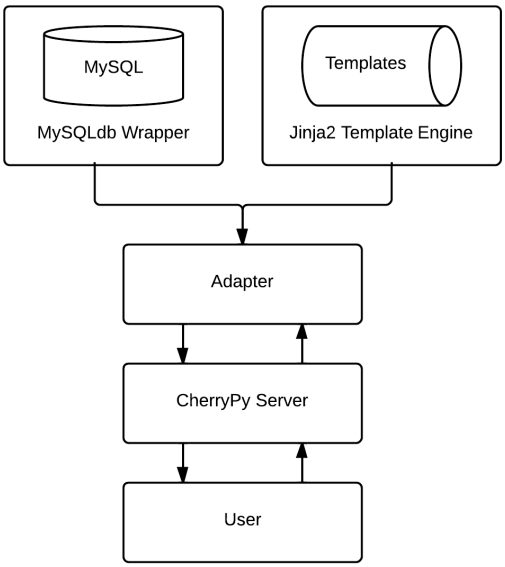

Figure 38: Web Interface Block Diagram

5.4 Hardware

The BeagleBone has relatively few external hardware components, consisting of a CAN transceiver, LCD, and a real time clock.

BeagleBone Breakout Cape

To make prototyping and troubleshooting easier we purchased a CircuitCo BeagleBone breakout cape. The cape features two cutouts, each equipped with a set of two 46-pin connectors. The 46-pin connectors in each cutout are wired in parallel to the other cutout giving us access to all of the signals into and out of the BB while a cape or circuit is connected. The breakout cape is shown in Figure 39.

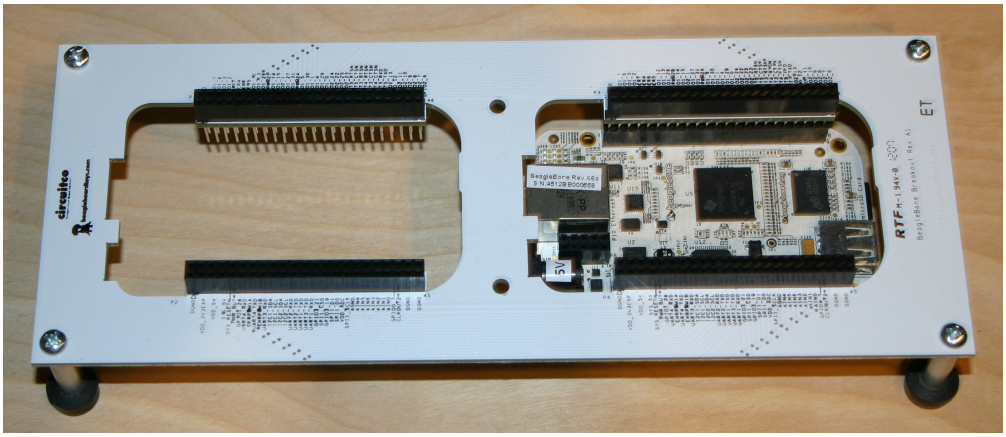

Figure 39: BeagleBone Breakout Cape

CAN Transceiver

The CAN transceiver for the BB was mounted on a small protoboard and plugged into one of the 46-pin headers. The 24V bus power is connected to the RJ-45 jack on the same protoboard.

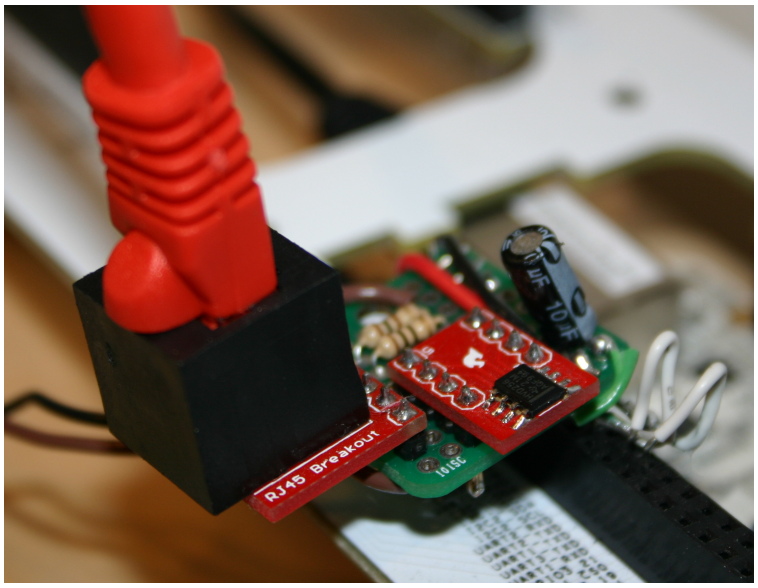

Figure 40: BeagleBone CAN Transceiver Prototype Implementation

LCD

For our touchscreen we chose the New Haven Display NHD-4.3-480272EF-ATXL#-T. This LCD has a 4.3 inch diagonal display with 480RGBx272 pixel resolution. It has a 3.3V 24 bit parallel interface and a 4

wire resistive touchscreen. We chose this LCD because of its low cost and its ability to operate on 3.3V logic. The display uses an integrated HiMax HX8257-A01 display driver which supports a 480RGBx272 pixel resolution. The physical BB to LCD connections can be found in Table 14. For prototyping we used the 40-pin FFC to through hole adapter shown in Figure 41. Originally we planned on using the backlight driver contained in the BB to drive the LCD but quickly learned that we would need an external backlight converter to supply the 32mA @ 19.2V needed. For prototyping we chose to power the backlight from a bench power supply until we achieved a stable image.

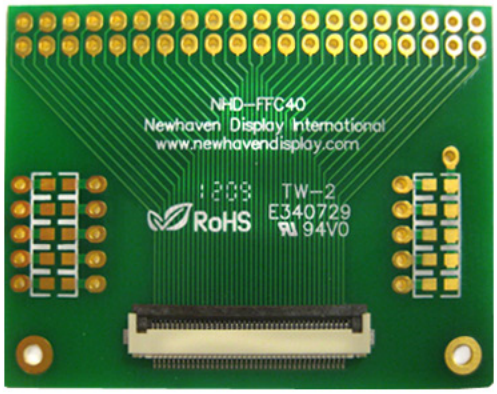

Figure 41: LCD Through Hole Adapter

For our first attempt at interfacing the LCD with the BB we used a breadboard as a patch panel, seen in Figure 42.

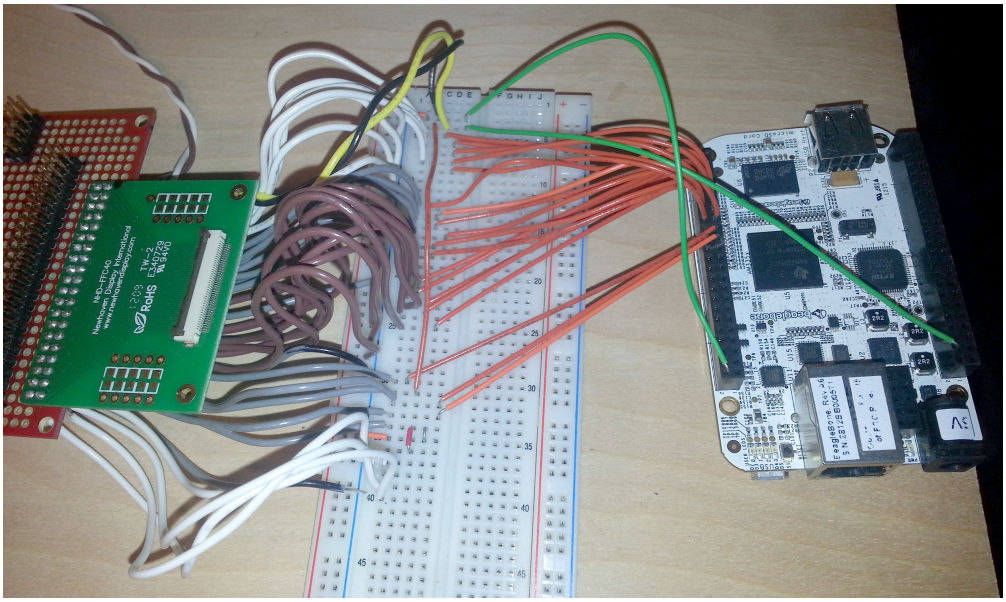

Figure 42: LCD Breadboard Connection

When we could not achieve a stable image we connected the LCD directly to the BB, to eliminate the breadboard from the equation, seen in Figure 43.

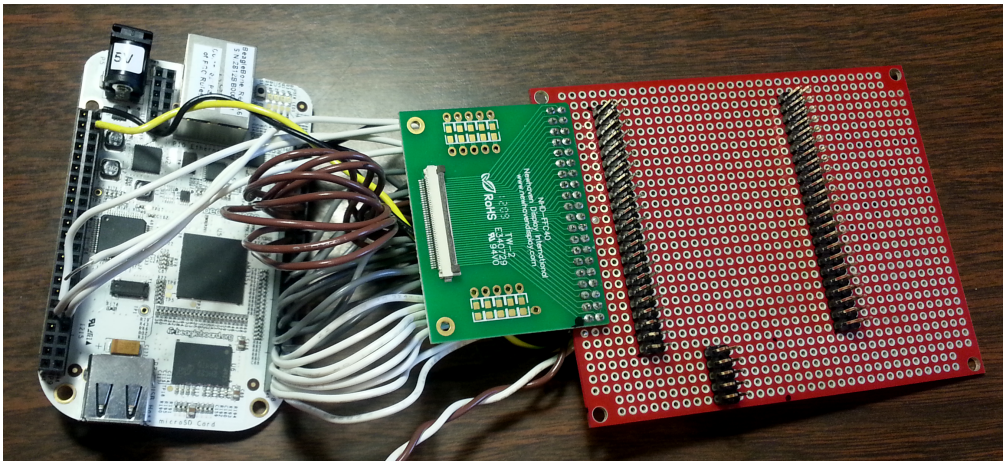

Figure 43: Direct LCD Connection

Once we confirmed that our connections to the LCD were correct we constructed a soldered interface which could be plugged directly into the BB, seen in Figure 44.

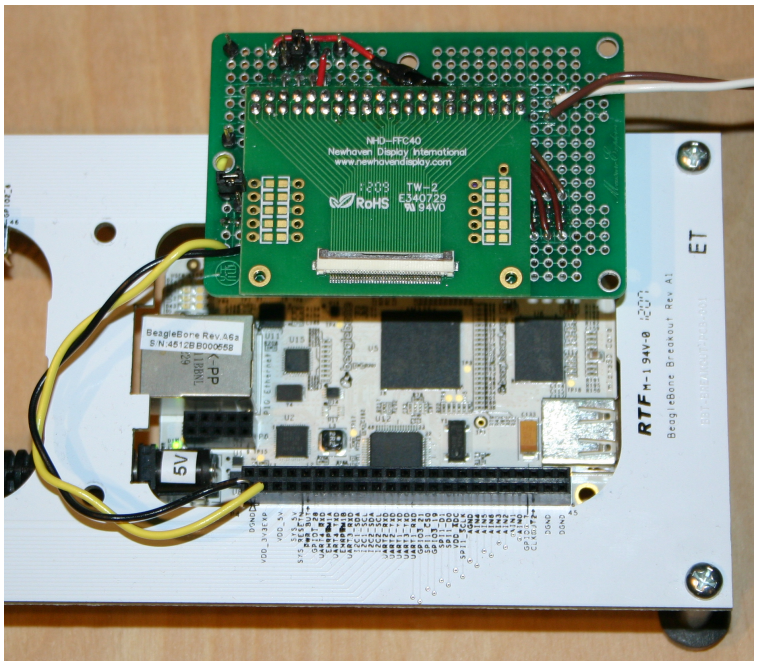

Figure 44: Soldered LCD Interface

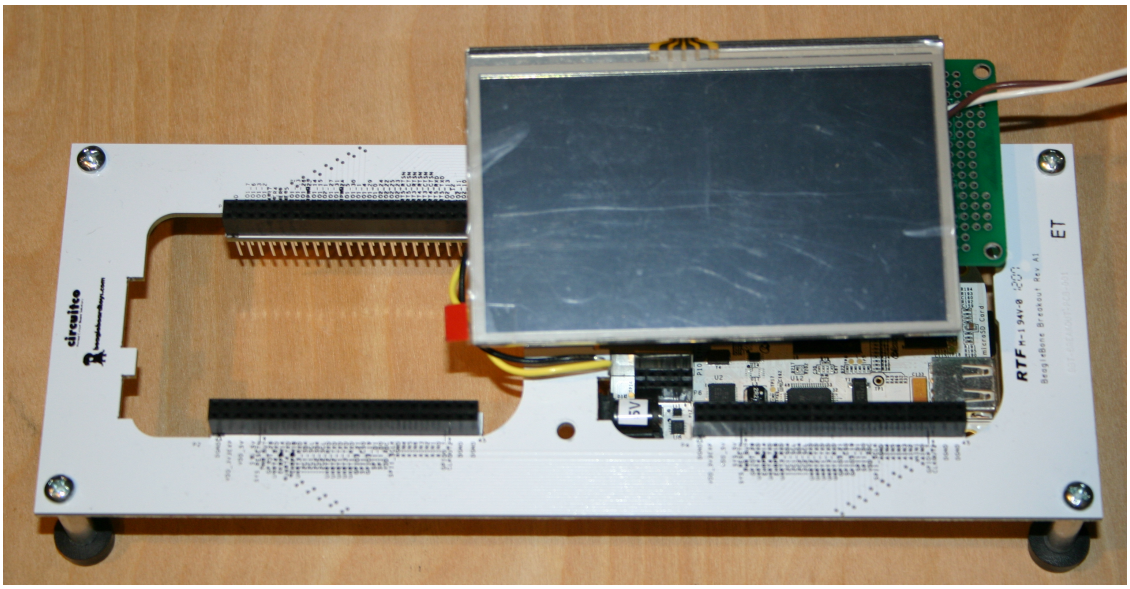

Figure 45: LCD Testing Prototype

RTC

The BB has no persistent real time clock, in the event of a power outage the system time will be affected negatively. If the BB is connected to the internet it will update the time from a central server; however, this is unreliable. It is important to maintain accurate time data even with power outages so we chose to add a real time clock with battery backup. We chose the Maximum Integrated DS3231S#. The DS3231 is an extremely accurate I^2C RTC with an integrated temperature compensated crystal. The DS3231 uses a 20mm 3V CR2032 coin cell which has an average capacity of 225mAh. This could power the DS3231 in timekeeping mode for more than 30 years.

6 PIC

For our remote nodes a microcontroller is needed to handle sensor data, outputs, and communications with the master node. To reduce part count and design complexity we opted for a microcontroller with an integrated CAN controller. The microcontroller we chose to handle communications between the BB and sensors or outputs is the PIC18F25K80. We chose this chip because it was the lowest cost option with an integrated Enhanced Controller Area Network (ECAN) module. The PIC is a 8-bit microcontroller with 32KB of program memory and 3 648 bytes of RAM. The integrated master synchronous serial port (MSSP) module is capable of both I^2C and SPI serial data communications offering compatibility with many different types of serial sensors. Additionally the PIC offers up to 24 pins of digital I/O and an eight channel 12-bit ADC making the PIC a powerhouse of interconnection for the development of sensor and control nodes. For software development we used Microchip's PIC integrated development environment MPLAB X in conjunction with the MPLAB C18 compiler and the PICkit3 in-circuit serial programmer (ICSP) to compile and load programs on to the chip. The PIC is tasked with three main things; communicating with sensors and EEPROM over I^2C; communicating with the master node (BB) over CAN; and setting the state of outputs based on messages from the BB.

6.1 Hardware

In all of our networked modules we utilized a PIC18F25K80 microcontroller which from this point on will be referred to as the PIC. The PIC is powered directly from the 5V rail of the buck converter. The PIC requires a $10k\Omega$ resistor from MCLR to V_{DD}, a $10\mu F$ capacitor from V_{DDCORE} to V_{SS} and a $0.1\mu F$ bypass capacitor from V_{DD} to V_{SS} for basic operation.

To allow for in-circuit programming we added a five pin programming header where the PICkit3 can be connected to each node. For the programmer to function we connected MCLR, PGD, PGC from the PIC to the programming header. Additionally we connected V_{DD} and V_{SS} to the header. When connected, the PICkit3 allows for programming and debugging of the PIC. We also added a heartbeat LED to an output allowing for visual confirmation that the firmware on the PIC is running properly.

Figure 46: PICkit3 Programmer

Buck converter

We chose the LM2671-5.0 because of its low cost, high efficiency, and wide input voltage range. Following the procedure contained in the datasheet we made our external parts selection. For the inductor we referenced Figure 47.

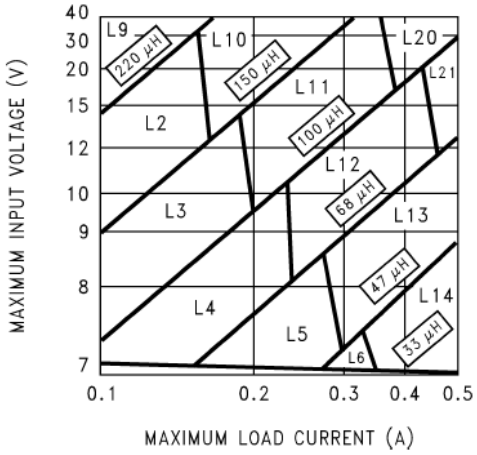

Figure 47: PIC Buck Converter Inductor Selection Chart

Looking at the chart with a 24V input and 500mA load it calls for L21. From the corresponding table in the data sheet we retrieved that we need an inductor with an inductance of $68\mu H$ and a current rating of 0.99A. We picked a Pulse Engineering PE-53821 powered iron toroid core inductor for its low cost and low EMI. The PE-53821 is no longer produced, we used the lead-free equivalent PE-53821NL which has an inductance of $78\mu H$ and a current rating of 0.82A. While the current rating is lower than the original inductor its higher impedance makes the lower current rating acceptable.

We chose a Nichicon UPJ1V121MPD1TD for our output capacitor. It is a $120\mu F$ electrolytic capacitor

with a voltage rating of 35V. It is rated for a ripple current of 550mArms at the switching frequency of 260kHz and has a low ESR of 0.22Ω. The PJ series of capacitors offers low impedance and high reliability making them well suited to switching power supplies in mission critical applications.

The catch diode needs to have a a voltage rating greater or equal to $1.25 \times V_{In}$ and a current rating of at least twice that of the load. We chose a 1N5158 Shockley diode to act as our catch diode. It is rated for 30V and 1A.

Following the datasheet we chose an input capacitor with a voltage rating greater than $1.25 \times 24V = 30V$ and with an RMS current rating greater than half that of the DC load, 250mA. We chose a Nichicon UPW1V151MPD. It is a $150\mu F$ electrolytic capacitor with a voltage rating of 35V and an RMS current rating of 555mA. The PW series offers low impedance, our capacitor has an ESR of 0.117Ω making it well suited for use in a switching power supply.

The buck converter also requires a $0.01\mu F$ 50V ceramic boost capacitor for proper operation. Assuming continuous conduction the calculations for duty cycle and ripple voltage are shown below. We extracted an operating efficiency of 87% from Figure 48.

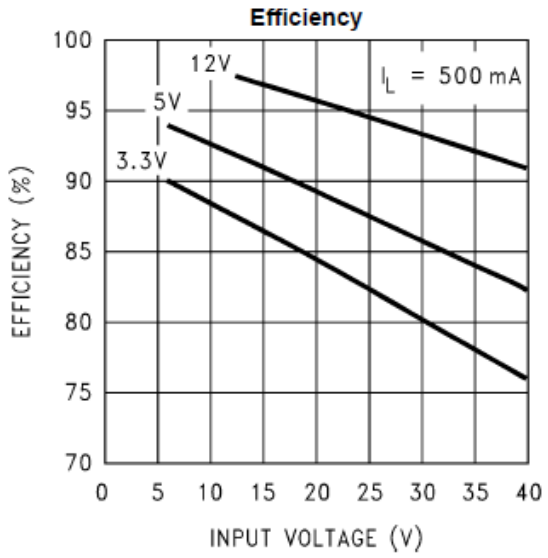

Figure 48: PIC Buck Converter Efficiency

Duty Cycle:

$$D = \frac{V_{OUT} \times \eta}{V_{IN(max)}} = \frac{5V \times 87\%}{24V} = 0.18125$$

Inductor Ripple Current:

$$\Delta I_L = \frac{(V_{IN} - V_{OUT}) \times D}{f_s \times L} = \frac{(5V - 24V) \times 0.18125}{260kHz \times 78\mu H} = 0.1698A$$

Output Ripple Voltage:

$$\Delta V_{Out} = \frac{T \times \Delta I_L}{8C} = \frac{(\frac{1}{260kHz})(0.1698A)}{8(120\mu F)} = 680\mu V$$

Ripple due to the ESR of the output capacitor:

$$\Delta V_{Out(ESR)} = ESR \times \Delta I_L = 0.22\Omega \times 0.1698A = 37.4mV$$

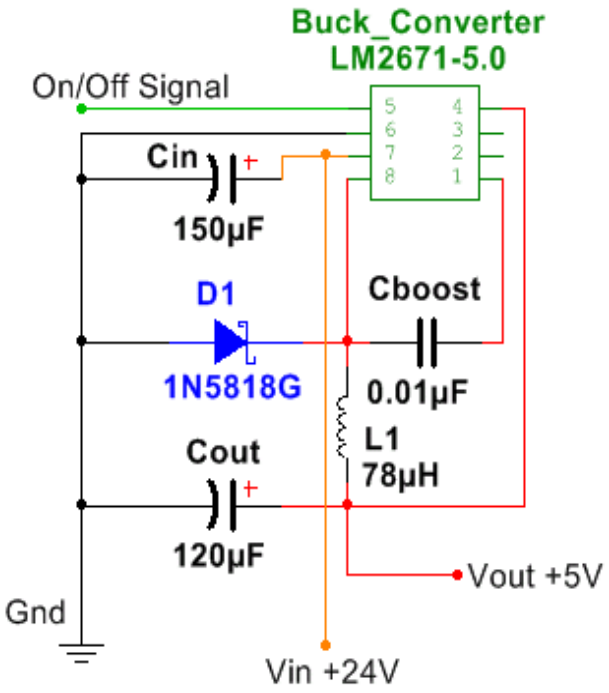

Figure 49: PIC Buck Converter Schematic

CAN

The CAN transceiver is connected directly to the CAN controller in the PIC. All of the input pins of the PIC are 5V tolerant so the V_{RXD} pin of the transceiver is connected to the 5V rail. For our purposes we have no need to change the transceiver to silent mode so we tied the mode select lead to ground. The transceiver is powered directly from the +5V rail supplied by the buck converter with a 0.1uF bypass capacitor. A 120Ω termination resistor is added to the last node on the network to prevent signal reflections in the cabling.

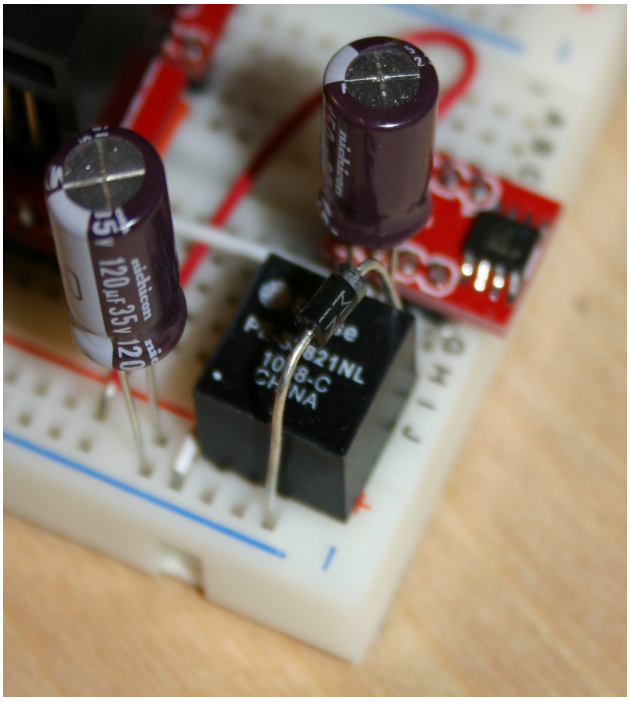

Figure 50: Buck Converter Prototype Implementation

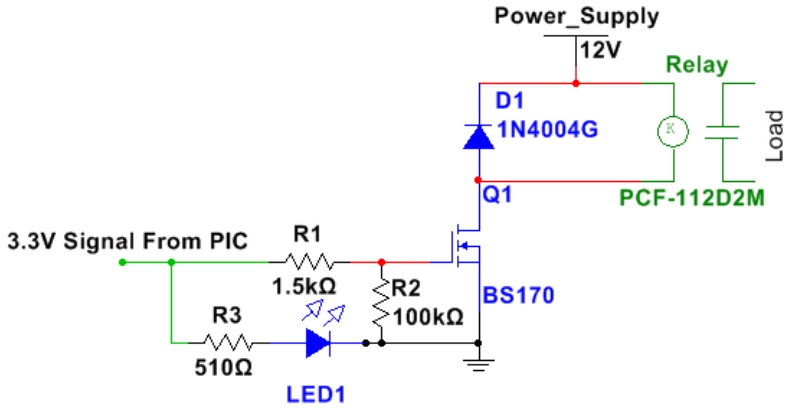

Figure 52: PIC CAN Transceiver Schematic

I^2C

The LM92 temperature sensor, the HIH-6131 Humidity Sensor, the K30 CO_2 sensor and the EEPROM all communicate over I^2C to the PIC. All of the sensors were chosen for their ability to run off of a 5V supply. The K30 CO_2 sensor depends on an internal microcontroller for calibration and data processing. The internal

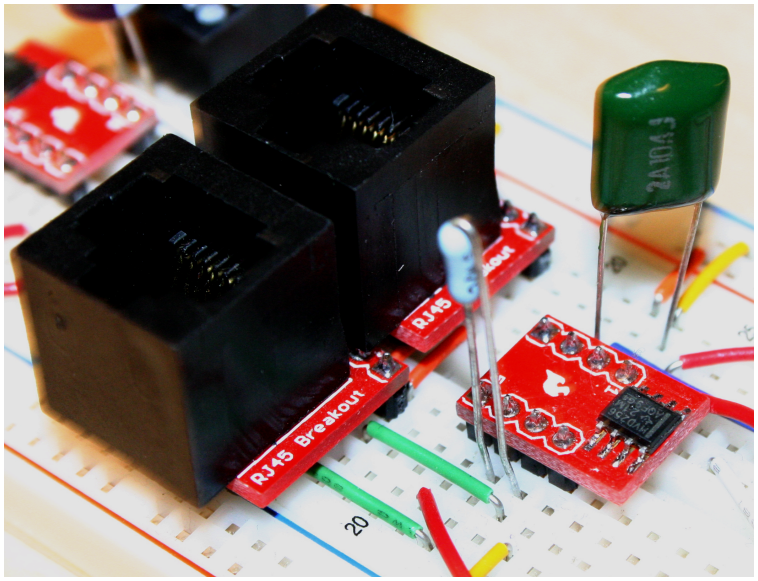

Figure 51: CAN Transceiver Prototype Implementation

microprocessor is only 3.3V tolerant limiting our I^2C data lines. Additionally the K30 limits the maximum speed of the I^2C bus to 100kHz.

Microchip 24LC00 EEPROM The 24LC00 is a 128-bit serial I^2C EEPROM. We chose the Microchip 24LC00 because it accepts supply voltages between 2.5V to 5.5V, has a low cost, and is readily available. The EEPROM will store any persistant data including unique node information such as CAN address, node type, sensors available and outputs available. It will also hold manufacturing information like model, serial number, software version number, and manufacture date.

6.2 Sensor Node Hardware Implementation

In addition to the PIC, the buck converter, and the CAN transceiver, the sensor node contains a 3.3V regulator and sensors for temperature, relative humidity and CO_2 concentration. The breadboard prototype for our sensor node is shown in Figure 53. In addition to the breadboard prototype we designed and assembled a printed circuit board for our sensor node.

Microchip TC1107 Linear Voltage Regulator In order for the LM92 and the HIH-6131 to properly interpret the 3.3V logic on the I^2C lines they must also be powered from a 3.3V rail. We implemented a Microchip TC1107-3.3 fixed voltage linear regulator to provide this rail. Together the LM92, HIH-6131 and EEPROM consume a maximum of 4mA, well within the 300mA maximum of the TC1107. A linear regulator was chosen over a switching regulator because it reduces external part count and noise. The only

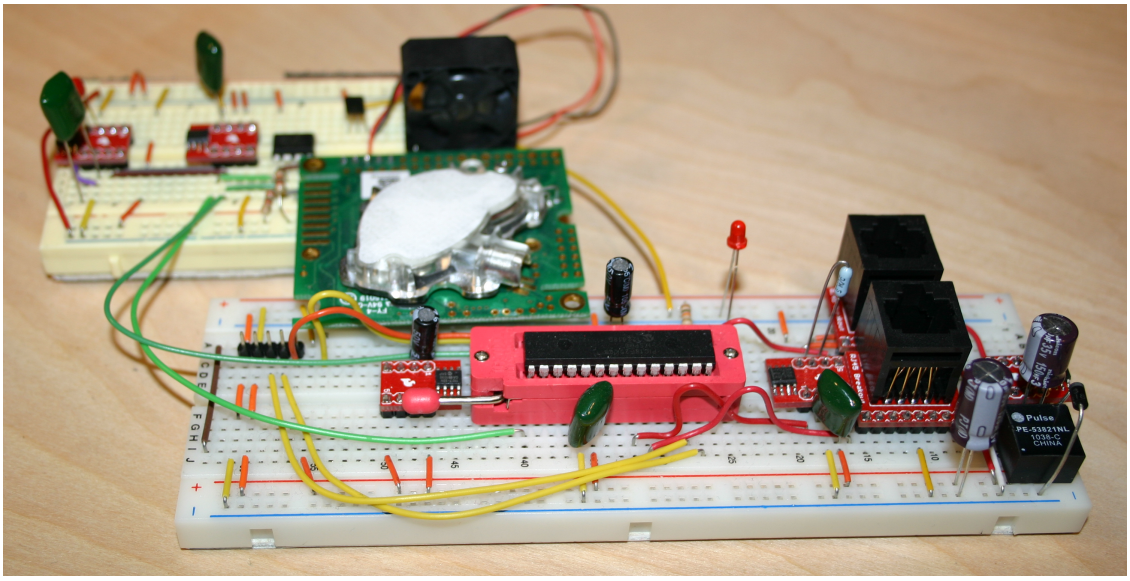

Figure 53: Sensor Node Breadboard Prototype

extra part the regulator requires is a $1\mu F$ compensation capacitor. The linear regulator dissipates 6.8mW when regulating 5V from the buck converter to 3.3V.

Texas Instruments LM92 Temperature Sensor The LM92 is connected to the I^2C bus and is powered from the 3.3V rail supplied by the TC1107. The interrupt and critical alarm outputs were unneeded for our application and were left unconnected. The address pins A0 and A1 were both pulled low giving the LM92 an I^2C address of 1001000. It also has a $0.1\mu F$ bypass capacitor.

Honeywell HIH-6131 Humidity Sensor The HIH-6131 is connected to the I^2C bus and is powered from the 3.3V rail supplied by the TC1107. The high and low alarms were unneeded for our application and were left unconnected. Both a $0.22\mu F$ bypass capacitor and a $0.1\mu F$ capacitor from V_{CORE} to V_{SS} are needed for proper operation.

Figure 54: HIH-6131 Humidity Sensor

SenseAir K30 CO_2 Sensor The K30 is connected to the main sensor node board via a five pin female header. The header contains the I^2C bus 5V and ground the last pin is connected to the on-board digital supply of the K30 and is left unconnected on the header.

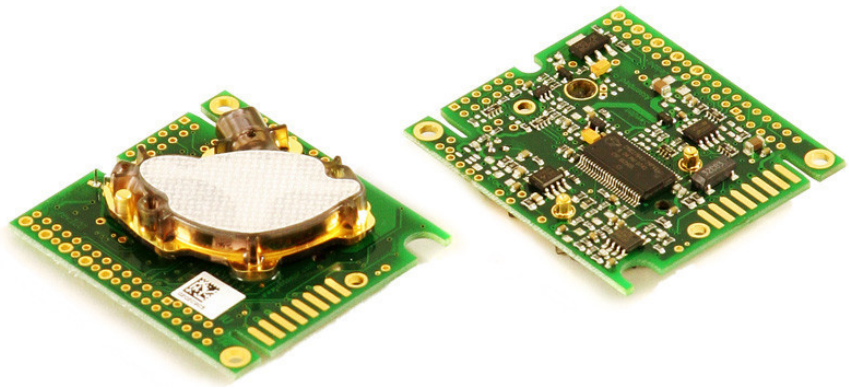

Figure 55: SenseAir K30 CO_2 Sensor

Microchip 24LC00 EEPROM The 24LC00 is connected to the I^2C bus and is powered from the 3.3V rail supplied by the TC1107.

Pull-up Resistors Both the I^2C clock (SCL) and data (SDA) lines are pulled up to the 3.3V rail by $2k\Omega$ resistors.

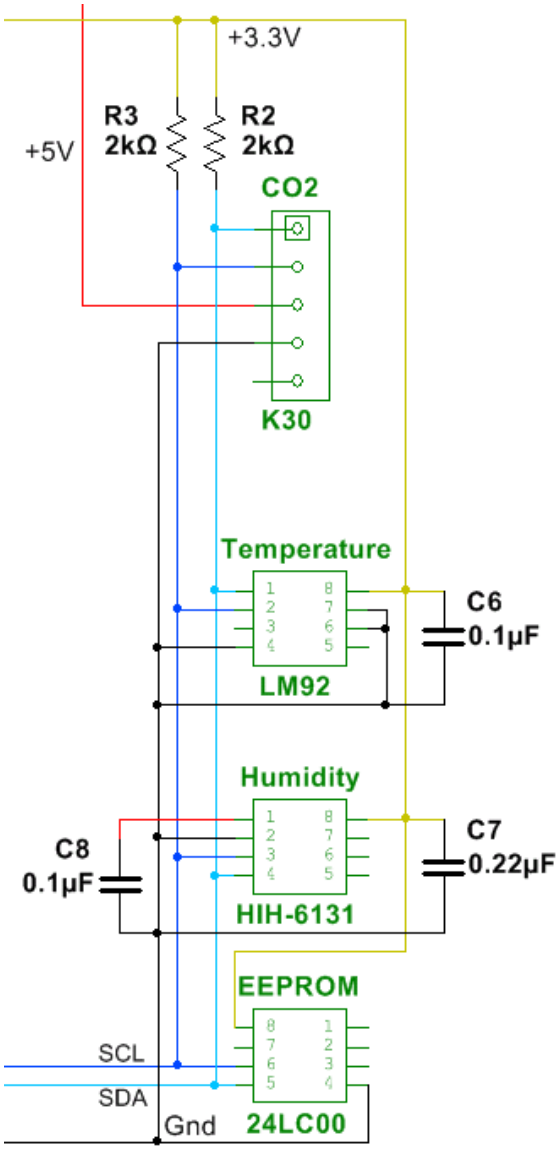

Figure 56: PIC I^{2}C Sensor Node Schematic

Fan For our sensors to provide responsive and accurate environmetal readings while enclosed in a partially sealed enclosure, forced ventilation is required. The volume of the sensor node enclosure is roughly 0.02 cubic feet. We chose a 25mm Copal F251R-05L3B brushless 5V fan. The fan draws 40mA and moves 1.2 cubic feet a minute, at this rate the air in the sensor node is exchanged with the surrounding environment every one second. The PIC is not able to source the 40mA required by the fan so we drove it using a BS170 MOSFET. By exchanging the air in the sensor node we reduce errors caused by enclosure heating due to radiant heat and decrease the sensor response time allowing for more accurate environmental control.

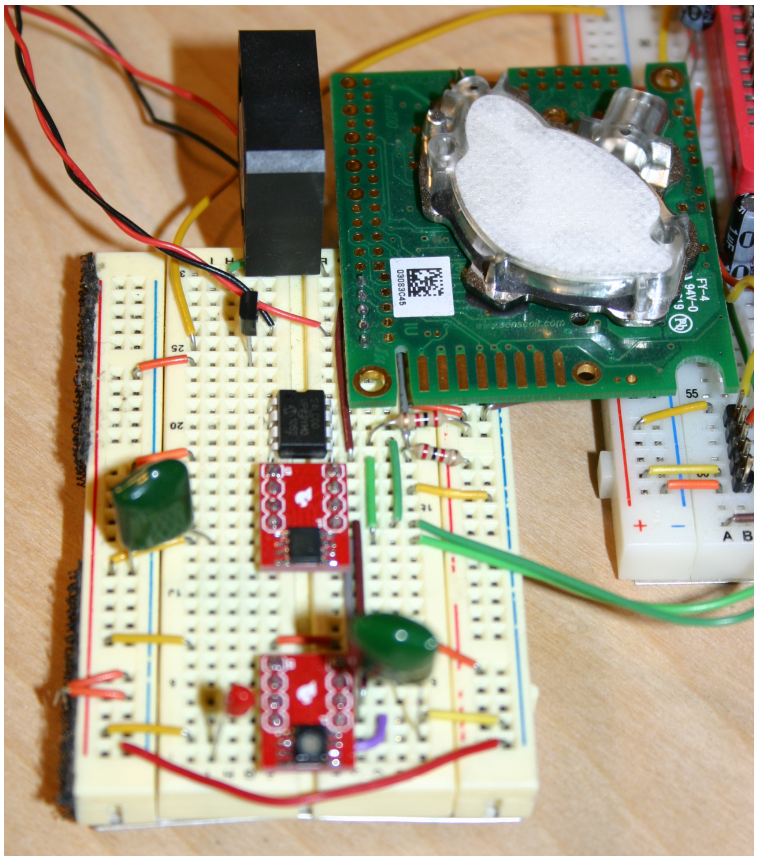

Figure 57: Prototype Sensor and Fan Implementation

6.2.1 Printed Circuit Board Design

We designed a printed circuit board for the sensor node. The primary design consideration for the sensor node was size. We wanted the sensor node to be as small as possible while still offering access to test points and, the expansion and programming headers. At 2" by 2.25" the largest component we had to consider in our board design was the K30 CO_2 sensor. Originally we considered a design in which the K30 was mounted directly above an identically sized board containing the other hardware, this would have resulted in a more compact design but at the cost of accessibility and airflow. We instead opted to use a design in which the K30 is mounted on a larger lower board. By using a larger lower board we were able to add test points for important signals to ease debugging. Additionally through-hole pads were added for unused pins on the PIC allowing future development without redesigning the PCB.

Mounting holes are provided on the four corners of the board. The two RJ-45 jacks are positioned directly next to each other in the north-west corner of the board below the mounting hole. The two jacks are wired in parallel allowing the network to pass-through the node. Rather than using discrete LEDs for node status information we chose to use jacks with integrated green and yellow LEDs.

Figure 58: RJ-45 Jack with LEDs

By using jacks with integrated LEDs we reduce the complexity of the enclosure design by requiring only one hole for both interconnection and indication. The resistors for the LEDs are positioned east of one jack and south of the other. The footprints for the jacks are positioned so that they extend past the edge of the board so that once installed in an enclosure the release tabs for the cabling can be easily depressed. A large copper area is used for the +24V bus power pass-through to minimize losses due to node insertion. A large copper trace connects the bus power from the RJ-45 jacks to the input of the buck converter.

The input capacitor for the buck converter is positioned in the north-west corner to the east of the mounting hole directly on the trace between the jack and the buck converter. A test point for the bus voltage and the ON/OFF signal to the buck converter are provided south of the input capacitor. The catch diode is positioned north-center and the buck converter is positioned directly south of the diode. The inductor is in the north-east corner west of the mounting hole. The output capacitor is situated directly south of the inductor. Large traces are used for all of the connections between parts of the buck converter. The +5V rail for the board is provided by a large trace which travels north-south on the top layer of the board. The 5V trace continues to the header for the K30 and an I^2C expansion header.

The CAN transceiver is east of the jacks and south of the buck converter. The CAN transceiver was positioned to minimize trace lengths between the jack, the transceiver, and the PIC to minimize noise. The 120Ω termination resistor can be soldered between the transceiver and the jacks in a vertical orientation. The PIC is almost centered south of the transceiver. Test points are provided on the east and west sides of the PIC for debugging and future expansion. The programming header is in the north-east south of the mounting hole. The area surrounding the header is left free of other components allowing the programmer to be plugged directly into the board.

The K30 is positioned south of the jacks leaving most of the test points around the PIC exposed. Two mounting holes are provided for standoffs for the K30. The LM92, the HIH-6131, the EEPROM and the 3.3V regulator are all positioned under the K30 when it is mounted. The I^2C lines were kept away from power lines to minimize noise. An expansion header is provided just south of the K30 offering connections to the I^2C lines, 5V, 3.3V and ground. A ground pour is used on both sides of the board to reduce noise.

Project information was added in copper on the bottom of the board and node information including serial, model and revision number were added in silkscreen on the top. The fan header was positioned in roughly the center of the board allowing the fan to be mounted on any side of the board. A 3D model of the finished board without the K30 installed is shown in Figure 59. The final design resulted in a board 2.26" by 4.2" more than adequately compact for our application. The final board design is shown in Figure 60.

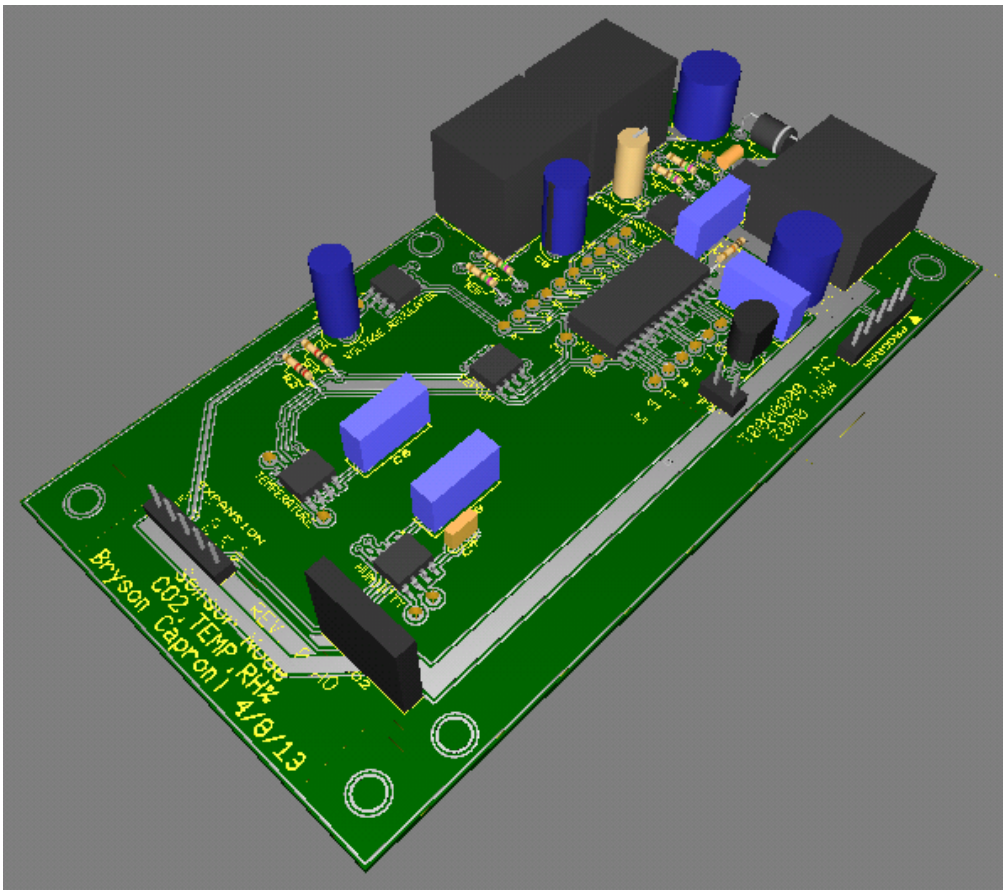

Figure 59: Sensor Node 3D Model

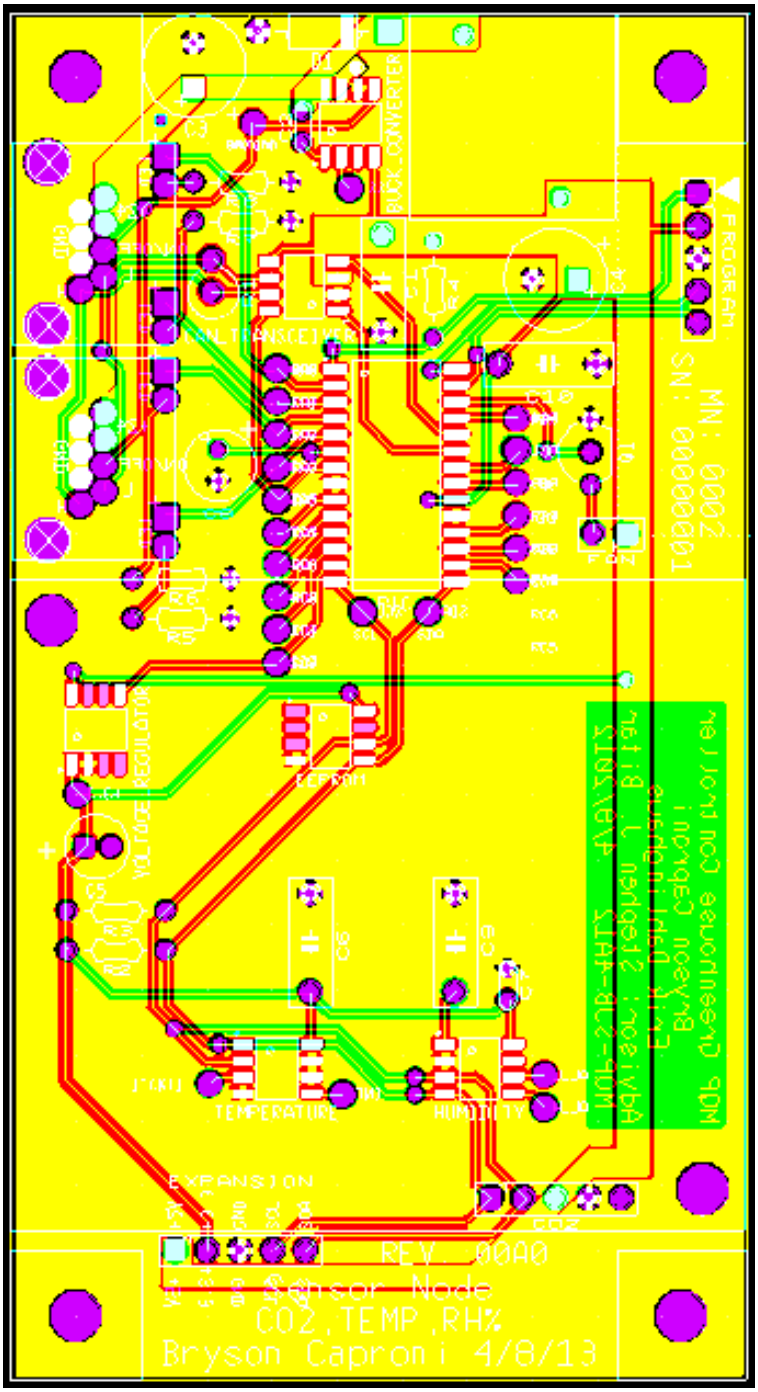

Figure 60: Sensor Node Printed Circuit Board Design

6.3 Control Node Hardware Implementation

The control nodes use the same PIC, EEPROM, buck converter and CAN transceiver as the sensor nodes making the basic construction almost identical. Where things begin to differ is that rather than sensors, control nodes contain circuitry that allows them to control high power loads. The block diagram for the control node is shown below.

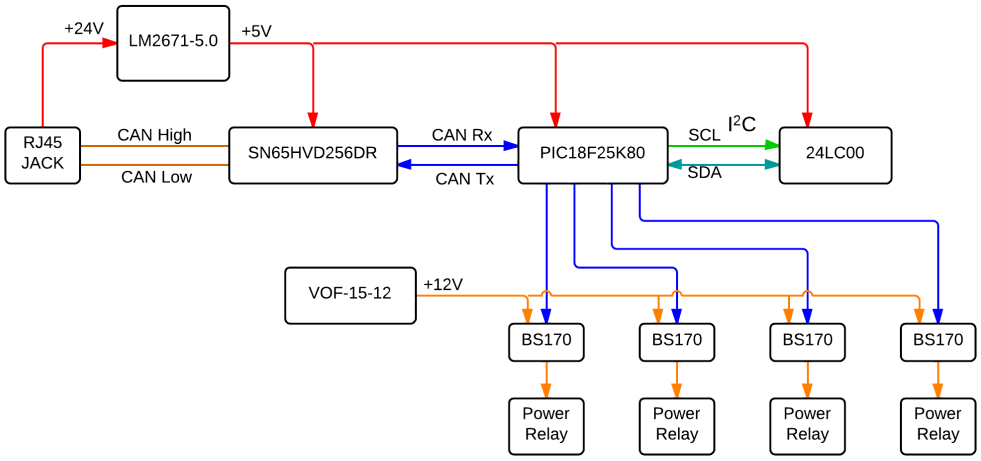

Figure 61: Control Node Block Diagram

The large pumps, lights, and motors in a green house present a challenging load to control using the PIC. We used power relays, however they are not without their challenges. Relay coils consume a considerable amount of power when energized, the relay we chose to use consumes 900mW of power. With a 5V coil this is a 180mA load; using the same buck converter as the sensor nodes a control node would be limited to two outputs. Many systems will need more than one set of outputs which increases cost and at around 2.5W for each control node this begins to present a considerable load on the bus. Unlike the sensor nodes who's power consumption may peak at 1W for a few milliseconds the control nodes place a continuous load on the network as long as the relays are energized.

To solve the power problems faced with using multiple bus powered control nodes we constructed a control node with more outputs and an on-board power supply to power the relays. With the bus power limitations removed the control node could be scaled up to have as many as fourteen outputs; for our prototype we built a control node with four outputs. We chose to use 12V as our relay coil voltage to reduce the current requirements in the driver circuit and provide compatibility with higher power contactors allowing future upgradeability.

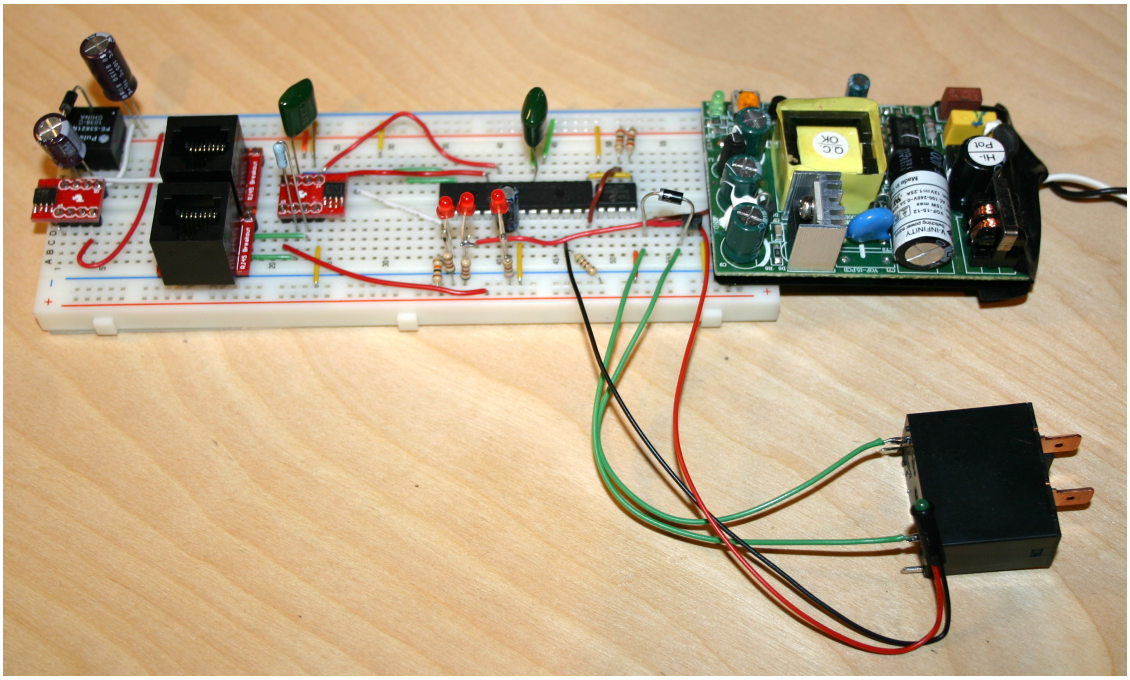

Figure 62: Control Node Protoboard Prototype

Control Node Power Supply

The control nodes still contain the same buck converter as the sensor nodes; it is used to power the PIC, the CAN transceiver and the EEPROM. By having the microcontroller and transceiver powered by the bus and not the on-board power supply it allows for the node to be reset using the node reset signal and the node can be powered up and configured without line power being applied to the control node. Additionally with the node powered from the bus line voltage monitoring could be easily implemented allowing the system to be aware of power outages or tripped circuit breakers. With line power already available in the control node for powering load devices we chose a CUI VOF-15-12 to energize the relays. The VOF-15-12 is a 15W 12V switching power supply with a wide input voltage range of 85-264 VAC and efficiencies up to 83%. With this power supply up to 14 relays could be added to the control node. Additionally it has no minimum load requirement.

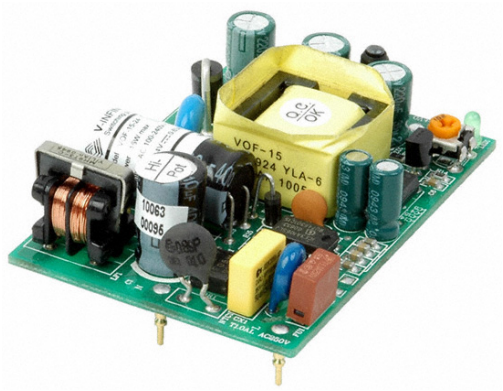

Figure 63: Control Node Power Supply

Power Relay and Driver Circuit

We chose a Tyco Electronics Connectivity PCF-112D2M power relay, it has a coil voltage of 12V and a load rating of 25A at up to 277VAC. We chose this relay because in addition to PC pin connections it offers 0.250" quick connects for the load allowing the use of high current wiring eliminating dependance on small PC board traces for power delivery.

Figure 64: Power Relay

The PIC cannot directly drive the power relay. We used an output of the PIC to drive a MOSFET which acts as a switch to drive the coil of the power relay. At 12V our relay requires 75mA to operate. We chose a BS170P MOSFET to drive the relay. The BS170 has a V_{DS} of 60V and a maximum continuous drain current of 270mA making it more than adequate for driving the coil of the power relay. Relay coils present a large inductive load, if the current flow through the coil is suddenly stopped, i.e. the BS170 is turned off, a build up of charge occurs causing a large voltage spike across the transistor. This voltage spike if left unchecked

could cause the MOSFET to fail. To quell the voltage spike we added a 1N4004 diode across the coil of the relay providing a path for the current stored in the inductor once the MOSFET turns off. The 1N4004 is a standard 1A 400V silicon rectifier diode. To provide over-voltage protection for the PIC in the event of a short from the 12V drain to the gate we added a $1.5k\Omega$ resistor in series with the gate. We added a $100k\Omega$ pull-down resistor to insure the MOSFET is completely off when the PIC is not driving it high. For more reliable operation we added a small green LED to indicate whether or not the relay is being driven by the PIC.

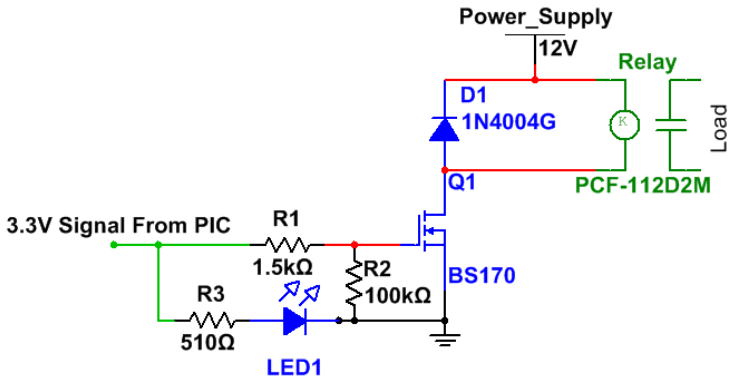

Figure 65: PIC Power Relay Driver Schematic

6.4 Software

The firmware on the PIC is very simple. When the PIC turns on it pulls its CAN address from the EEPROM and then sends a start message to the BB. Once the node starts it waits for a message on the CAN network from the BB telling it to either: poll the sensors and return the information; or set the state of an output. In order to reduce the amount of code, one program was developed for both the sensor and control node; its function can be switched by changing the NODETYPE definition to the node type. A flow diagram of the program can be seen in Figure 66. A complete source code listing is available in Appendix C on on page 105.

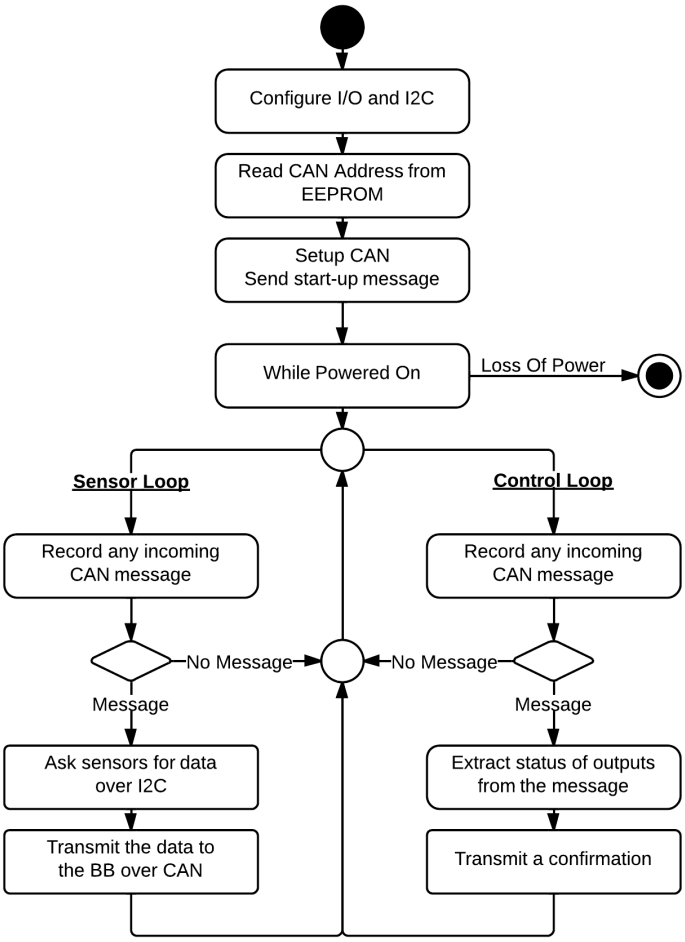

Figure 66: PIC Program Flow Diagram

7 Conclusion

We successfully completed six out of our original ten design goals; however, the unmet goals could be easily implemented. Both the float switch interface, and pH and electroconductivity sensors were left unimplemented due to time constraints, but could be easily integrated with minor additions to the existing node hardware and software. The touchscreen was successfully interfaced with the BeagleBone, however we were unable to successfully display a desired image. In our prototype phase we were able to achieve over 70% stability, an estimate taken from a five week period of testing. A summary of our design goal acheivements can be found in Table 12.

Goal	Result
100% Stability	70%
Power Failure State Resume	Success
127 Networked Nodes	Success
Interface via Touchscreen	Interfaced, but unsuccessful
Interface via Web	Success
One-Cable Interface	Success
Temperature, RH%, CO_2 Sensors	Success
pH and EC Sensors	Unimplemented
Float Switch Interface	Unimplemented
Output via Switched Outlets	Success

Total Parts Cost	$406

Table 12: Summary of Acheivements versus Design Goals

We have demonstrated that a full featured greenhouse controller for controlled environment agriculture can be produced at a low enough price-point to make it an attractive option for small startups and hobbyists. Our system is a practical alternative to existing solutions; the computer controlled system offers logging capabilites, remote access, and modularity allowing for future growth. While there are similarly featured products costing upwards of $3 000, most products at our price-point are severely lacking in functionality and often rely on mechanical timers and controls for operation. Full page photos our of final breadboard and solder board prototypes can be found on the following pages in Figure 67 and Figure 68.

It is clear that a drastic change to our food supply system must be made in order to make it more sustainable. Controlled environment agriculture may be the solution, but it will depend on the successful development of robust, effective control systems. Companies like PodPonics and The Farmery give us a glimpse into the potential future of agriculture, offering innovative solutions to the problems which face our food system today; however, low cost solutions are needed to reduce the barrier to entry into controlled environment agriculture. Inexpensive control systems like ours will pave the way, allowing companies and ideas like these to flourish.

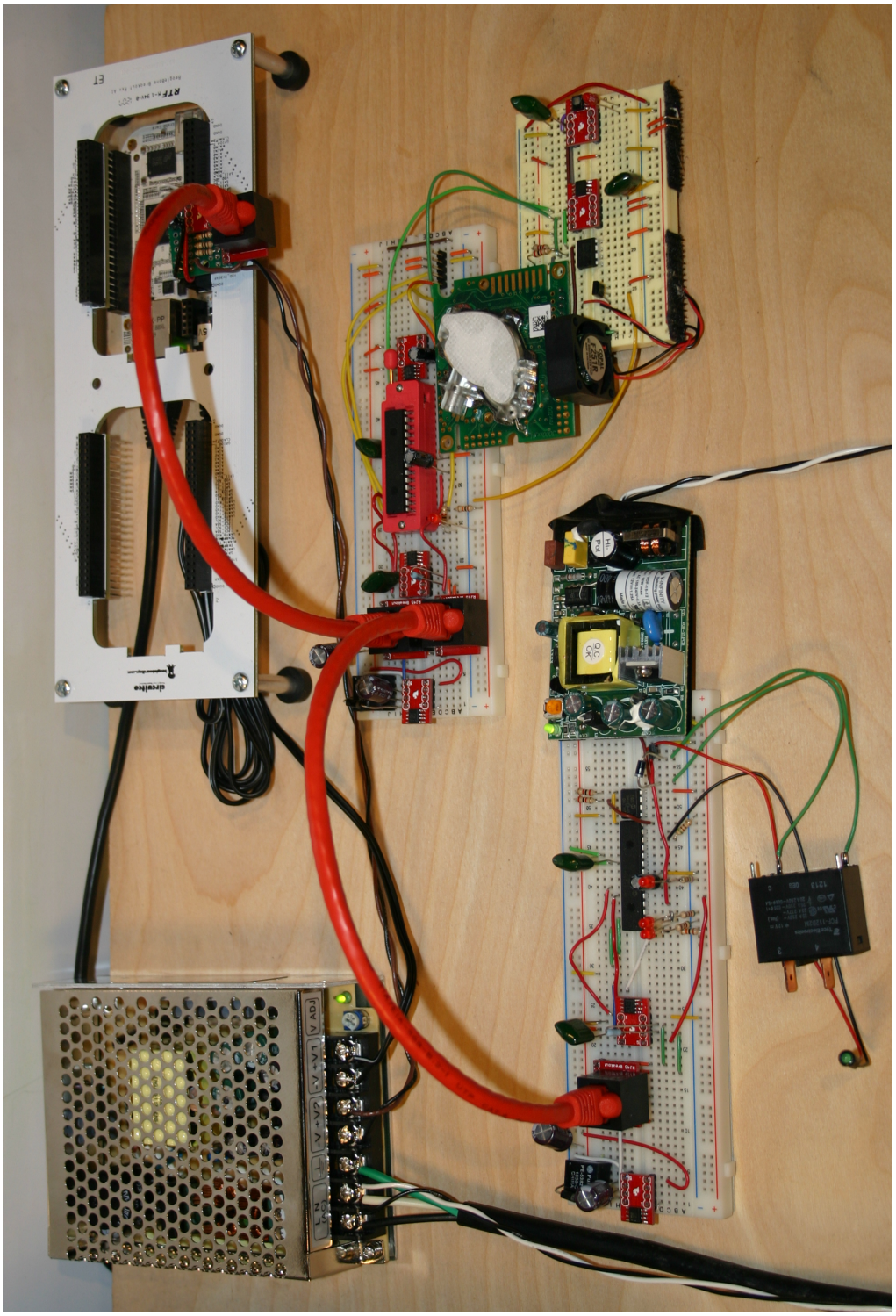

Figure 67: Breadboard Prototype - Full Resolution

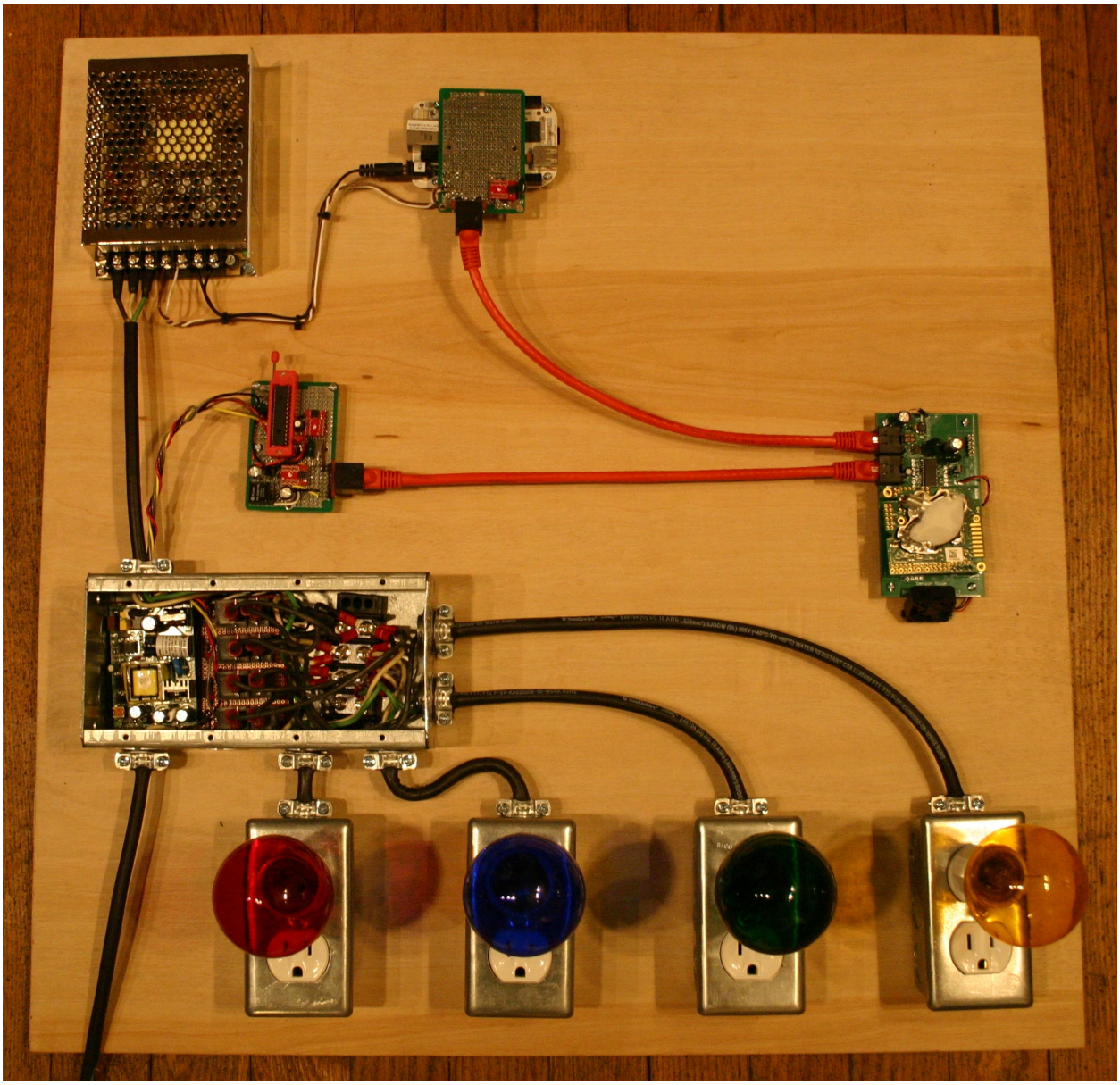

Figure 68: Solder Board Prototype - Full Resolution

7.1 Results

These sections describes the results we experienced when testing our final prototype.

7.1.1 CAN Network

Once the CAN modules were configured on the BB and the PIC, the CAN network preformed robustly.

CAN Message Confirmation We connected an Aligent MSO6012A mixed signal oscilloscope to monitor CAN tranmissions. We connected the analog inputs to the CAN bus signals; CAN high is shown in green and CAN low is shown in yellow. Triggering on a serial bus signal with a traditional oscilloscope is very difficult, luckily the MSO6012A offers triggering on CAN version 2.0A and 2.0B signals. By setting the approiate baud rate and selecting channel 2 as the CAN_H signal we achieved a stable signal display. In addition to monitoring the bus levels we used the digital inputs of the MSO6012A to monitor the Tx and Rx lines of the different nodes. The first pair is the BB; D0 is Tx and D1 is Rx. The second pair is the sensor node; D2 is Tx and D3 is Rx. And the final pair is the control node; D5 is Tx and D6 is Rx. The diagram showing the CAN message flow is shown in Figure 32. The oscillograms of the CAN messages are shown in Figure 69 through Figure 72.

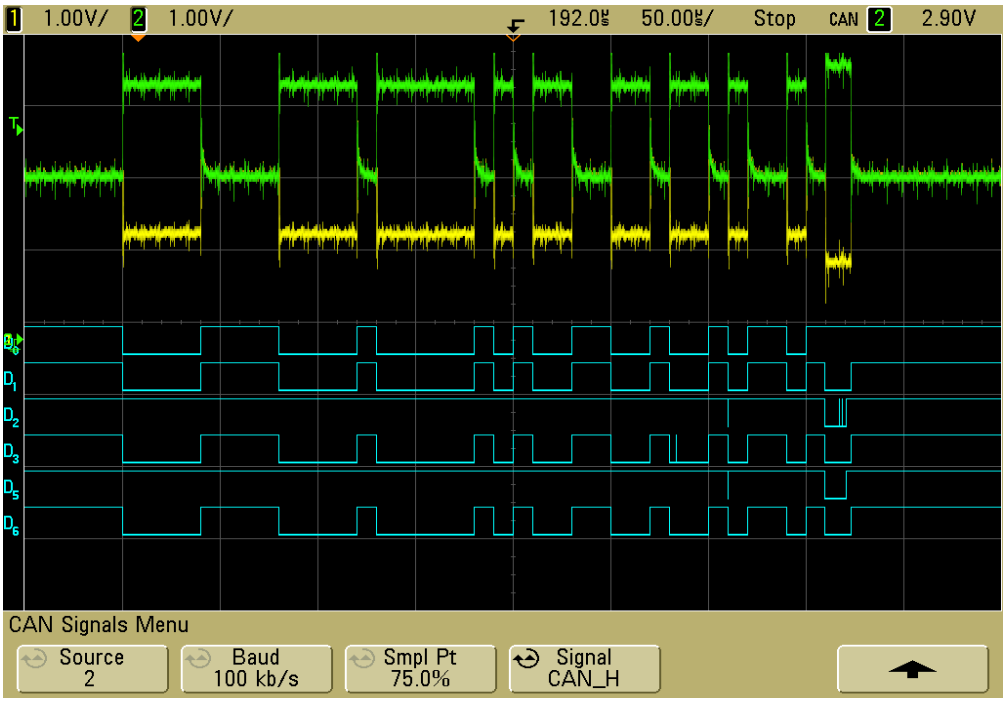

Figure 69: BB–>Sensor Node: Request for Data

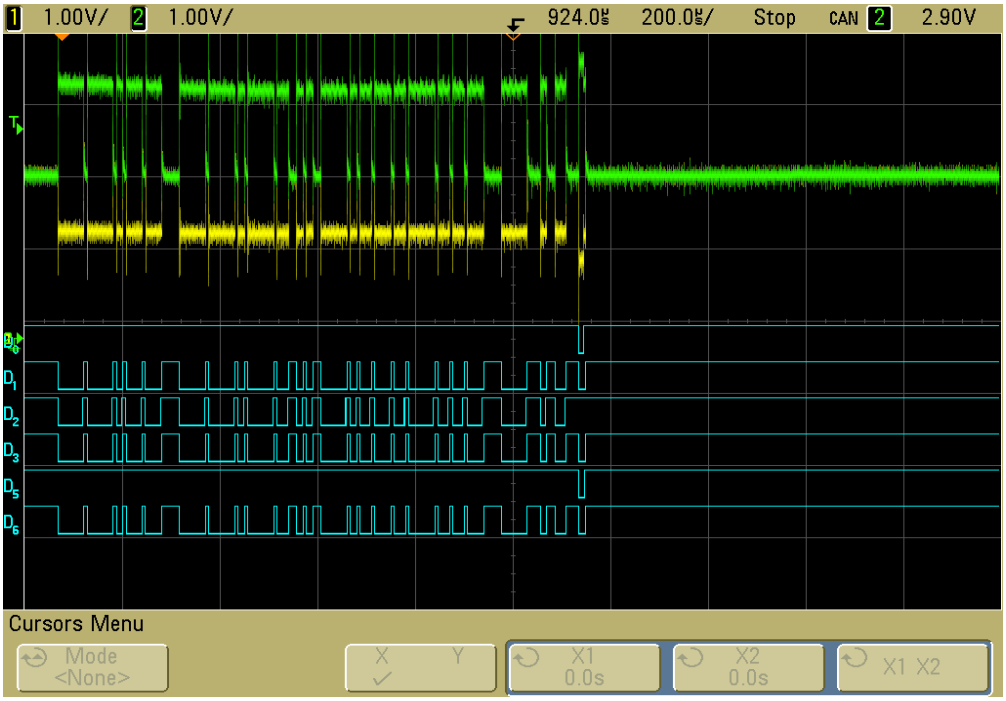

Figure 70: Sensor Node–>BB: Return Data

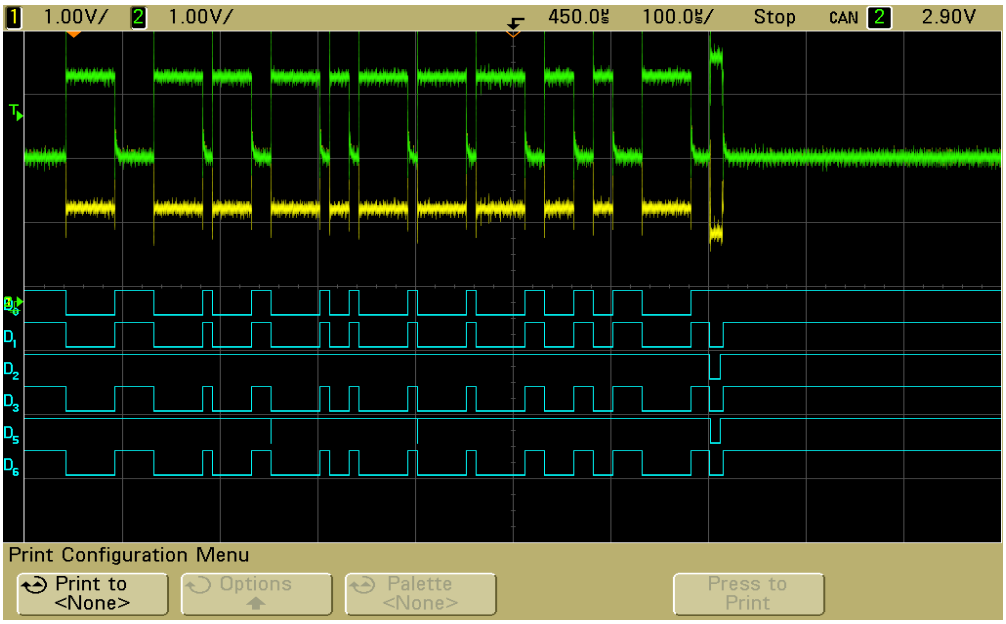

Figure 71: BB–>Control Node: Send Control Configuration

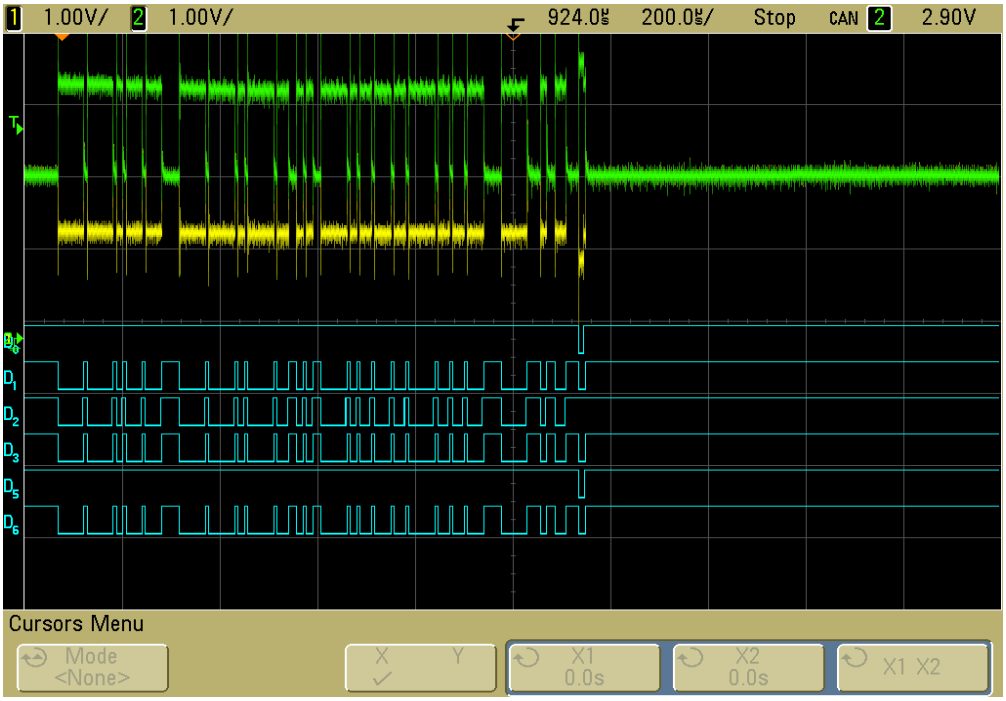

Figure 72: Control Node–>BB: Acknowledge Control Configuration

7.1.2 Hardware

I²C Signal Confirmation We connected an Aligent MSO6012A mixed signal oscilloscope to monitor I²C signals between the PIC and the sensors in the sensor node. We connected both the analog and digital inputs to the I²C bus lines. The I²C clock line SCL is shown by analog channel 1 in yellow and digital input D8. The I²C data line SDA is shown by analog channel 2 in green and the digital input D9. In addition to having the ability to trigger on CAN messages the MSO6012A offers extensive triggering functions for I²C messages. Triggering was accomplished by selecting channel 2 as the SDA line and channel 1 as the SCL line. The MSO6012A offers the ability to be triggered by an I²C frame with a specific address and data value. By setting the address to match the address of the sensor of interest and setting the data value to 0xXX the scope is triggered by any message to that address making capturing the fast paced massages between the PIC and the sensors much easier. The I²C oscillograms for the temperature, humidity and CO_2 sensors are shown in Figure 73 through Figure 75.

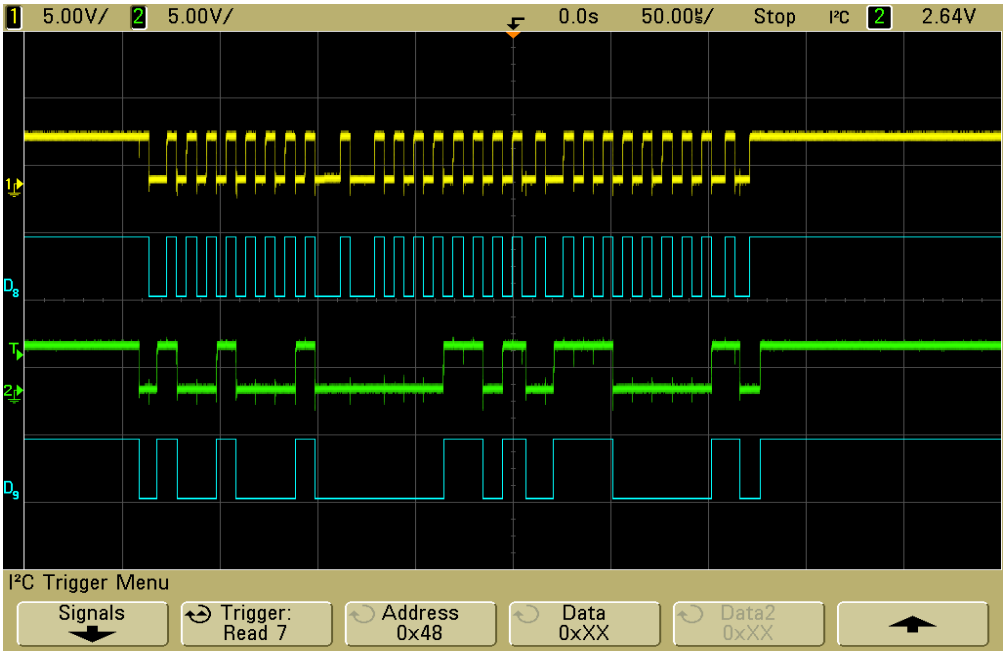

Figure 73: I^{2}C Temperature Sensor Signal

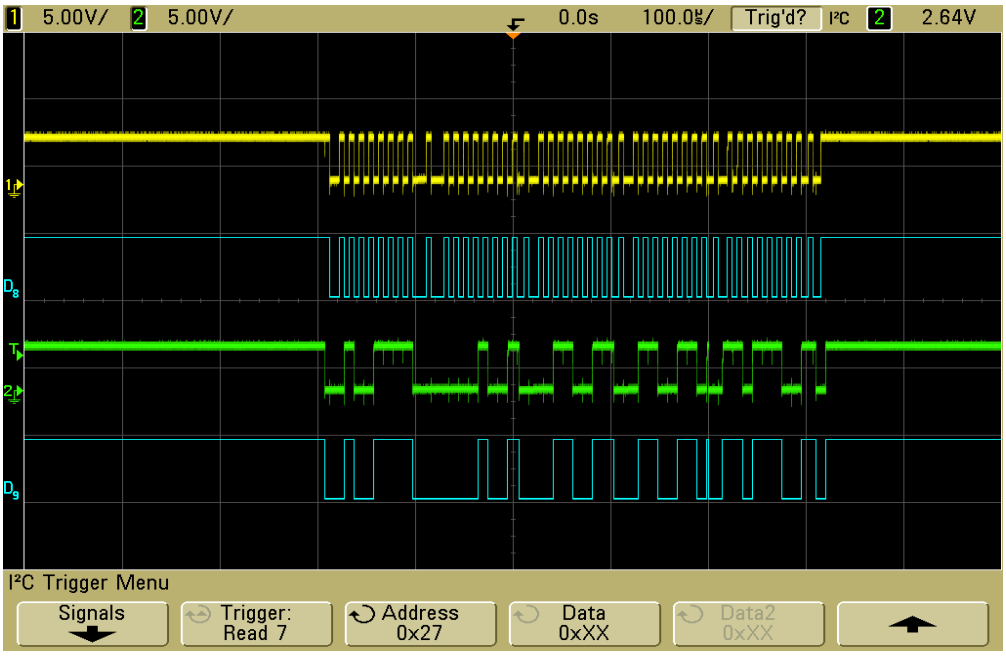

Figure 74: I^{2}C Humidity Sensor Signal

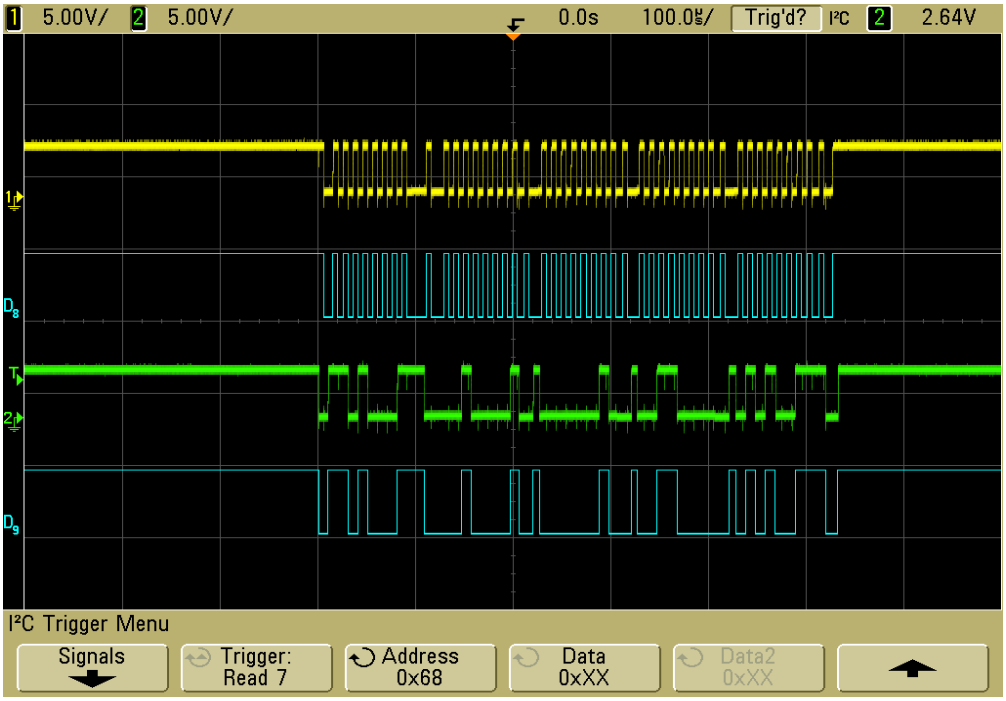

Figure 75: I^{2}C CO_2 Sensor Signal

7.1.3 Software

Creating a software application of this size was no simple task. Despite this, we were quite successful at creating a working application; the only real shortcoming was stability. Also, due to time constraints, some parts of the web application were not implemented.

Stability

We intended on having 100% stability in our system. We were unable to acheive this goal because of time constraints; unable to debug the software properly. The software that runs on the BB is fairly complex, and depends on many packages. Often times trying to fix a simple depedency issue or attempting to modify a path would create problems elsewhere that may go unannounced for some time. Additionally the BeagleBone software CAN message handling is hazard prone; a better solution would be to write an application in C using the SocketCAN library to handle the function of message handling and database storage. The reason we chose not to do this was ease of impementation; as we had already developed an SQL interface in Python.

Web Page

It was originally our intent to make a full featured web application allowing the user to change all parameters of the system. This turned out to be a massive endeavor that proved out of our scope. Despite this we were

able to develop the framework for a dynamic web page, and created a demo page which displays the data from a specific node on the network.

Our database functioned, but as the table size increased with more data the queries became slower and slower. We're still not entirely sure whether this is a restriction of the hardware, or a problem with the way we're interacting with the database. Some solutions would be to use an offloaded SQL server, but this is undesirable because it requires a persistant internet connection and would require significant bandwidth. Instead using a faster database system such as SQLite could potentially solve the problem; or, SQL could be abandoned in favor of a file-based object database like Python's built in module *shelve*.

7.1.4 Unimpemented Goals

There were several design features from our original goals that we decided not to implement due to time constraints. This includes pH and EC sensors, as well as an interface for float switches. Originally it was planned to have a touchscreen; however we were unable to attain complete functionality.

Touchscreen

The 24-bit LCDC on the BeagleBone is theoretically capable of driving our LCD, however we were unable to acheive full functionality. The LCD would not display the image we placed on it, instead it would display a number of different things; most notably it either showed a test pattern of which we never found the source; or four small off color versions of the image we were attempting to display, localized to the top quarter of the screen. Figure 76 shows the image we placed on the frame buffer, Figure 77 shows how the LCD interpreted the data. We theorized that the problem concerned the timing settings in the driver; however we tried many settings and none resulted in desired operation. Figure 78 shows an oscillogram of the LCD timings; the pixel clock is shown in blue, the frame clock, or vertical sync is show in purple, and the line clock, or horizonal sync is shown in green. We verified with both the LCD manual, and the AM335x manual that these were in fact the correct timings. It could also be a hardware issue; unlike our implementation, all commercial LCD capes implement a buffer between the BeagleBone and the LCD. Reviewing the datasheets for both the BeagleBone and LCD we determined this was unnecessary and was forgone because it required a 32 channel buffer, making it difficult to prototype with. If a PCB was designed the buffer would be included to test this theory. Further evidence from the BeagleBone mailing list points to this as a possible solution.

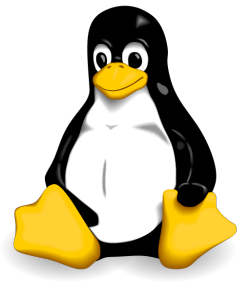

Figure 76: An Image of Tux

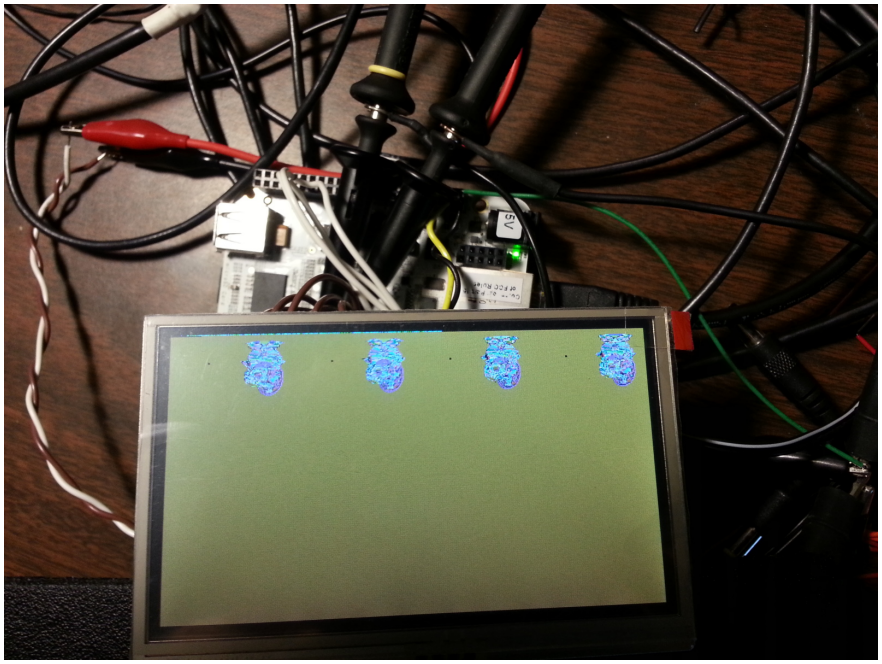

Figure 77: LCD Screen with Tux on Framebuffer

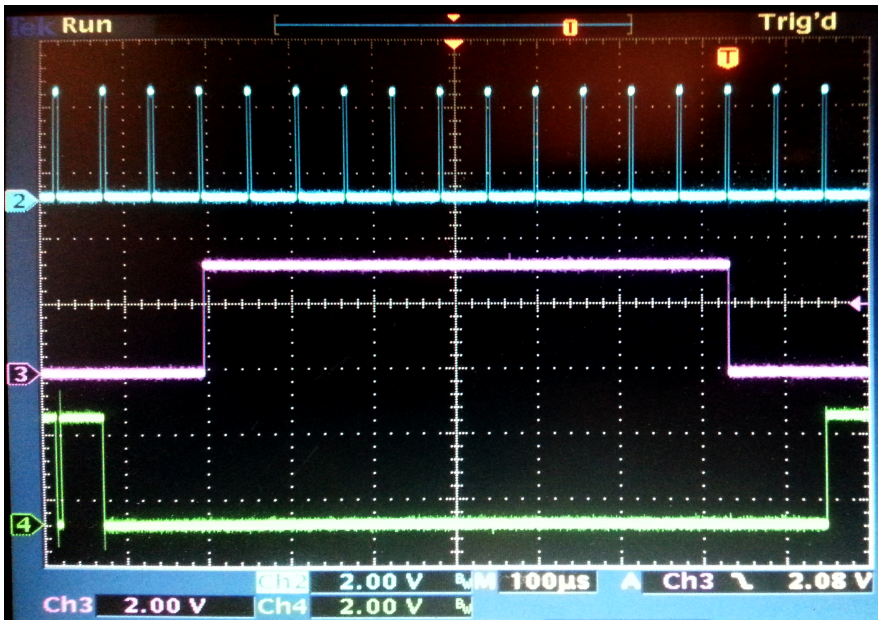

Figure 78: LCD Timing Confirmation

7.2 Recommendations and Future Work

As our project comes to a close there are many ideas that we have to further our work on this project to indeed create a market ready product. This section will discuss those things, or any other final remarks on the project.

The hardware section was written by Bryson Caproni. The software section was written by Erik Dahlinghaus.

Hardware

1. A PCB for the BeagleBone could be designed including a buffer between the LCD and the BB.
2. A PCB for the output node could be designed including current and line voltage monitoring for power consumption data.
3. The PCB for the sensor node could be redesigned with all SMD components to reduce size.
4. Additional sensor nodes could be created by interfacing other sensors including pH and EC.
5. The buck converter for each node could be redesigned for the current requirements to save cost.
6. An appropriate backlight driver for the LCD could be implemented allowing it to run off of the 5V supply of the BB.

7. Power supplies could be designed to replace existing supplies that were purchased to save on cost and size.

8. Extended performance testing of the CAN network could be done to determine the maximum number of nodes and cable length.

9. Crystal Oscillators could be added to all sensor and control nodes increasing network stability by increasing synchronization between nodes, allowing higher data rates.

Software

1. Firstly, if this project were to be redone or continued, someone with a more focused study in computer science would be integral to the success of the project.

2. If the project was to be started again I would probably redo the entire project in C instead of Python. I chose Python because I wanted to challenge myself and learn a new language. Python is an exceptional language, it is very simple to do things; there are Python packages for almost anything. However, C would operate much quicker, and then the SocketCAN library for C could be used. MySQL database interaction is equally as simple in C than in Python. Furthermore, the website could remain in Python, or be moved to a more traditional Apache-MySQL-PHP configuration.

3. Using a ready built ORM such as SQLAlchemy could solve some of the issues with the MySQL database lagging as the table size grew. This problem may be associated with some bad coding practices; however, I don't know SQL well enough to really know. SQLAlchemy encapsulates database access into functions and turns the data into convenient objects.

4. The system needs to be set up in order to accept new nodes on-the-fly. This could reasonably be accomplished. It would involve some sort of node announcement, followed by an address arbitration sequence (in case there was another node of the same address on the network), and then it could add itself to the database. After this the user would just assign the node into a zone.

5. Instead of holding the 'zone map' and 'environmental parameters' in an SQL database (or it could remain mirrored there, or imported/exported), a scripting language could be employed allowing users to create zone maps and environmental routines which they could share with other people. For instance, someone comes up with a tomato profile to grow tomatoes, which is very successful because of the temperature and lighting cycle, this person could then share this profile with other users.

6. The PIC code could also be drastically improved; instead of forwarding raw data from the sensors to the BB it could interpret the data first, and possibly make advanced decisions about it. Furthermore, it could be made possible so nodes could communicate with each other sans master node; although, this would have to be carefully executed to avoid causing uncontrolled havoc.

7. The PIC could be interrupted by the alarms built into the sensors to send important data directly to the BB (priority messages about TEMP_HIGH or other).

8. Subnets could be used to address all sensors simultaneously when requesting data.

9. Lastly, the web application needs significant work. With the framework that exists for it now it will be simple to implement a full featured control panel by which to control the system. This includes features like graphing, log export, 'zone map' import, a settings manager, and numerous pages which could show you data about the environment in interesting ways. Furthermore data could be converted to JSON objects and a simple JSON API could be published allowing other users to interpret data in their own ways.

Bibliography

[1] Kenya Nurse, "Hydroponic lettuce", 2012.

[2] Jean Smith, "Aeroponics", 2011.

[3] Crop King, "Small scale hydroponic growing", 2011.

[4] Ron Carroll, "Aeroponics systems", 2012.

[5] Indoorganics, "Backyard hydroponics", 2012.

[6] Tohxiu Jin, "Why hydroponics", September 2011.

[7] Geoff Wilson, "Aquaponics network australia", 2013.

[8] NY Sun Works, "The science barge", 2012.

[9] Aaron Saenz, "Transforming shipping containers into local farms - podponics brings produce to the city", August 2011.

[10] The Farmery, "Design of the farmery", 2012.

[11] Plantlighting Hydroponics, "Cap cgc-1e growroom controller", 2012.

[12] Colorado Aquaponics, "Aquaponics: Growing fish and plants together", 2013.

[13] Arroyo Instruments, "Thermistors", 2012.

[14] Microchip, "Temperature sensor design guide", 2012.

[15] Denes K. Roveti, "Choosing a humidity sensor: A review of three technologies", July 2001.

[16] All About Circuits, "ph measurement", 2012.

[17] Atlas Scientific, "ph sensor", 2012.

[18] Steve Corrigan, "Introduction to the controller area network (can)", Application Report SLOA101A, July 2008.

[19] softing, "Can bus bit-stuffing rule", 2013.

[20] Armin Bassemir Florian Hartwich, "The configuration of the can bit timing", 6th International CAN Conference, November 1999.

[21] Parthiban Kesavan, "Introduction to controller area network (can bus)", 2013.

[22] Leroy Davis, "Can bus pin out, and canopen pinout, with signal names:", February 2012.

[23] M. Tim Jones, "Anatomy of the linux kernel", 2013.

[24] Ohio State University, "Hydroponic greenhouse lettuce enterprise budget", April 2011.

[25] John Hopkins Center for a Livable Future, "Food distribution and transport", 2013.

[26] Julie Beaulac, "A systematic review of food deserts", July 2009.

[27] H. Charles J. Godfray, "The future of the global food system", September 2010.

[28] Merle H. Jensen, "Controlled environment agriculture in deserts, tropics, and temperate regions - a world review", 2013.

[29] James Clawson, "Developing a sterile environment for aeroponic plant growth", 2013.

[30] City Grow, "Aeroponic systems", 2011.

[31] Fadhrick Pickaso, "Commercial food productions and hydroponics", 2012.

[32] Aquaponic Secrets, "Aquaponic systems growing methods", 2011.

[33] Sylvan H. Witter, "Rising carbon dioxide is great for plants", Policy Review, Fall 1992.

[34] Aquarious Technologies, "Electrolytic conductivity measurement: Theaory & application", March 2002.

[35] Pedro M. Ramos, J.M. Dias Pereira, Helena M. Geirinhas, and A. Lopes Ribeiro, "A four - terminal water - quality - monitoring conductivity sensor", March 2008.

A List of Materials

Part	Manufacturer	Model	No.	Unit Price
BeagleBone	CircuitCo Electronics LLC	BB-BONE-000	1	$89
CAN Transceiver	Texas Instruments	SN65HVD256DR	3	$1.62
CAN Termination		120Ω 1%	2	
Touchscreen	New Haven Display	4.3-480272EF-ATXL#-T	1	$38
FFC Breakout	New Haven Display	NHD-FFC40	1	$10
RTC	Maxim Integrated	DS3231S#	1	$8.38
RTC Battery	Sparkfun Electronics	CR2032	1	$1.95
Battery Holder	4UCON Technology Inc.	Coin Cell Holder - 20mm	1	$1.50
Pull-up Resistor		4.7kΩ 5%	2	
RJ45 Jack	Assmann WSW Components	A-2014-2-4-N-T-R	4	$0.80
System Power Supply	CUI Inc.	VGD-60-D524	1	$38.93
Node Microcontroller	Microchip Technologies	PIC18F25K80-I/SP	2	$3.64
Vcore Cap		$10\mu F$ 25V	2	
Bypass Cap		$0.1\mu F$	6	
Buck Regulator	Texas Instruments	LM2671M-5.0	2	$3.28
Buck Inductor	Pulse Engineering	PE-53821NL	2	$3.24
Buck Cin	Nichicon	UPW1V151MPD	2	$0.48
Buck Cout	Nichicon	UPJ1V121MPD1TD	2	$0.54
Buck Diode	Micro Commercial Co	1N5818-TP	2	$0.42
Buck Cb	Kemet	C317C103K5R5TA	2	$0.24
3.3V Regulator	Microchip Technologies	TC1107-3.3VOA	1	$0.46
Compensation Cap		$1\mu F$ 25V	1	
EEPROM	Microchip Technologies	24LC00-I/P	2	$0.30
Temperature Sensor	Texas Instruments	LM92CIM	1	$4.06
Humidity Sensor	Honeywell	HIH6131-021-001	1	$21.79
Humidity Cap		$0.22\mu F$	1	
CO_2 Sensor	SenseAir	K-30	1	$65
Fan	Copal Electronics Inc	F251R-05L3B	1	$12.75
I^2C Pull-up		2 kΩ	4	
Control Power Supply	CUI Inc.	VOF-15-12	1	$16.80
Power Relay	TE Connectivity	PCF-112D2M,000	4	$4.58
MOSFET		BS170	6	$0.25
Relay Trigger LED	Lumex	SSI-LXH312GD-150	4	$1
Output Indicator Lamp	Arcolectric	LE2951WL5G	4	$5.88
SOIC-8 Breakout	Sparkfun	BOB-00494	6	$2.95
Heartbeat LED		Small Red LED	2	
LED Resistor		750Ω	6	
		Total:		$406

Table 13: Complete List of Parts

B Full Schematics

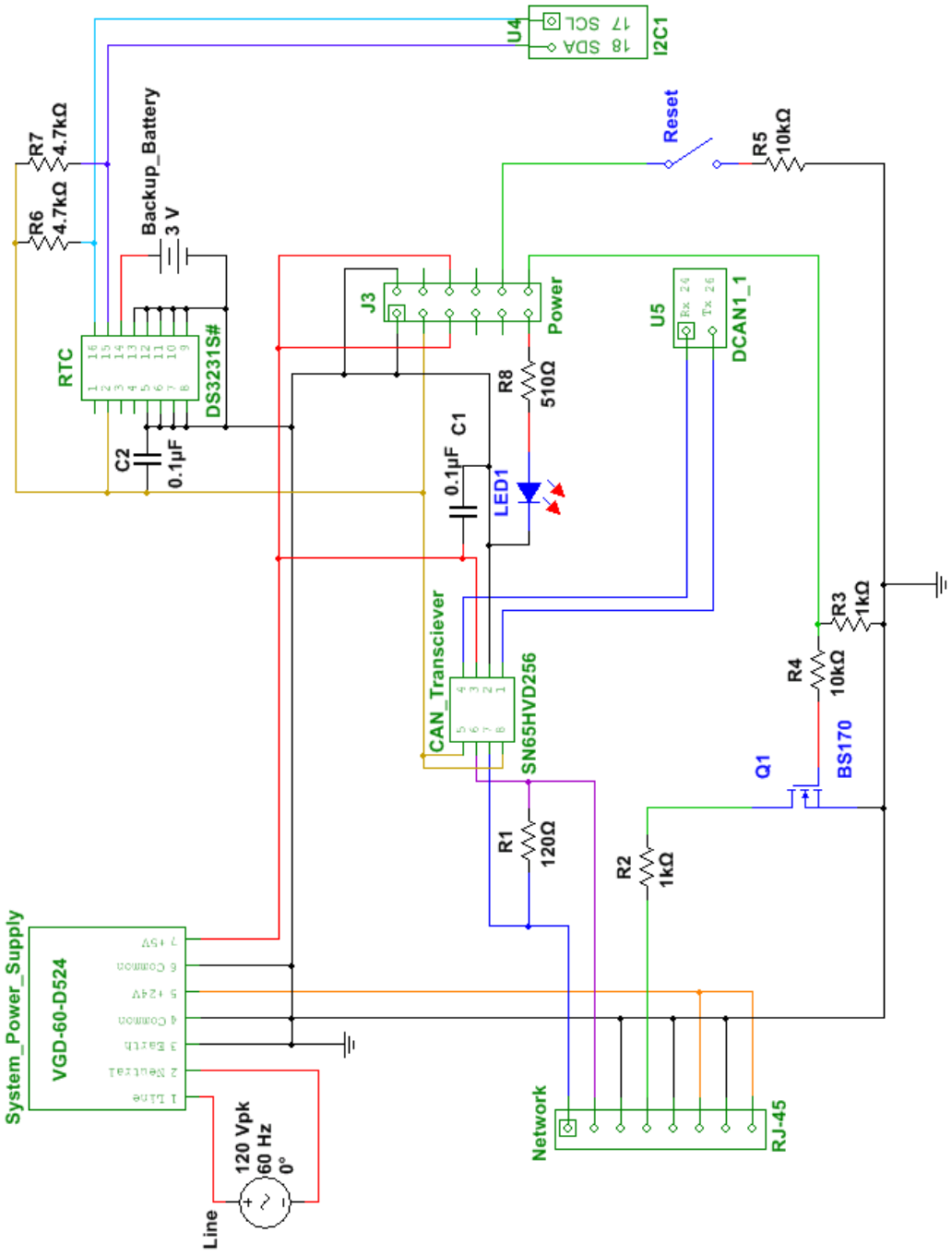

Figure 79: BeagleBone Schematic

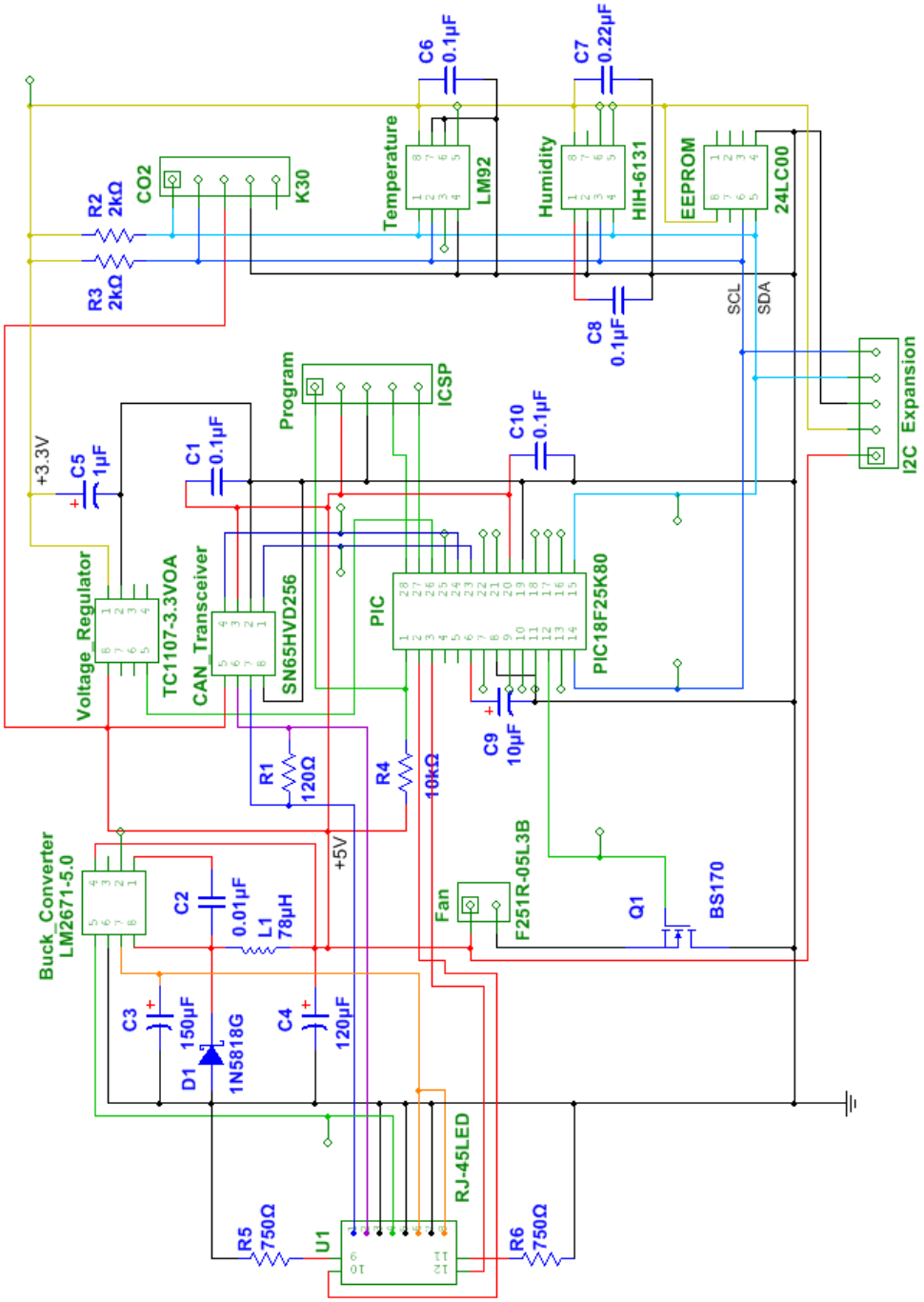

Figure 80: Sensor Node Schematic

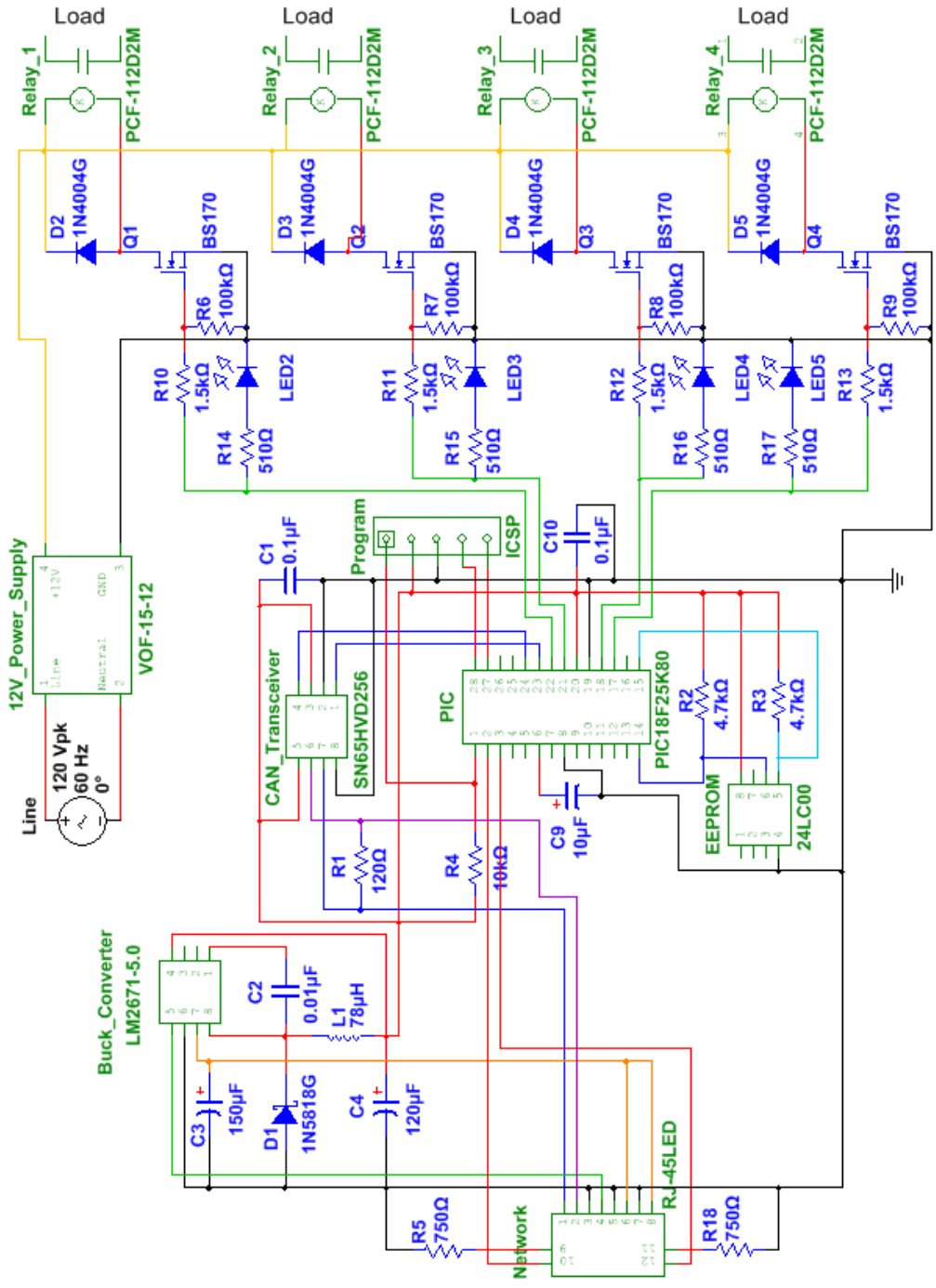

Figure 81: Control Node Schematic

LCD Pin	Signal	BB Pin	Signal	Description
1	LED-		Backlight Ground	
2	LED+		Backlight Supply	32mA @ 19.2V
3	GND		GND	Ground
4	VDD		LCD Power	+3.3V
5	R0	45	lcd_data0	Red Data
6	R1	46	lcd_data1	Red Data
7	R2	43	lcd_data2	Red Data
8	R3	44	lcd_data3	Red Data
9	R4	41	lcd_data4	Red Data
10	R5	42	lcd_data5	Red Data
11	R6	39	lcd_data6	Red Data
12	R7	40	lcd_data7	Red Data
13	G0	37	lcd_data8	Green Data
14	G1	38	lcd_data9	Green Data
15	G2	36	lcd_data10	Green Data
16	G3	34	lcd_data11	Green Data
17	G4	35	lcd_data12	Green Data
18	G5	33	lcd_data13	Green Data
19	G6	31	lcd_data14	Green Data
20	G7	32	lcd_data15	Green Data
21	B0	15	lcd_data16	Blue Data
22	B1	16	lcd_data17	Blue Data
23	B2	11	lcd_data18	Blue Data
24	B3	12	lcd_data19	Blue Data
25	B4	17	lcd_data20	Blue Data
26	B5	14	lcd_data21	Blue Data
27	B6	13	lcd_data22	Blue Data
28	B7	19	lcd_data23	Blue Data
29	GND		GND	Ground
30	PCLK	28	lcd_pclk	Pixel Clock
31	DISP	18	On/Off Signal	Active High
32	HSYNC	29	lcd_hsync	Line Clock
33	VSYNC	27	lcd_vsync	Frame Clock
34	DE	30	lcd_ac_bias_en	Data OK
35	NC			
36	GND		GND	Ground
37	XR	40 (P9)	Touch XNUR	
38	YD	37 (P9)	Touch YPLL	
39	XL	39 (P9)	Touch XPUL	
40	YU	38 (P9)	Touch YNLR	

Table 14: BeagleBone to LCD Interconnection

C Source Code

A complete, up-to-date copy of the source code is available via Github. This includes the full software package for the BeagleBone, the patches to the Angstrom kernel, and the full source code for the PIC microcontroller. Additionally further documentation and information about licensing can be found there.

https://github.com/ErikDahlinghaus/greenhouse-controller

The software for this project is available under the GNU GPLv3 License.

http://www.gnu.org/licenses/gpl-3.0-standalone.html

Listing 1 shows the directory structure of the software package for the BeagleBone.

```
/monitor.sh
/daemon.py
/lib
     /__init__.py
     /classes.py
     /pid.py
     /sql.py
     /zone_runnable.py
/startup
     /log
     /start.sh
/can
     /log
     /start.sh
     /candump
     /handler.py
/web
     /log
     /start.sh
     /controller.py
     /templates
          /index.html
          /nav.html
          /pagetemplate.html
     /static
          /style.css
          /jquery.js
```

Listing 1: Software Directory Listing